Making Diagnosis Explicit

Kathleen M. KING

Ph.D.
University of Edinburgh
1993

Declaration

I declare that this thesis has been composed by myself and that the work described is my own.

Kathleen M. KING

Abstract

Thesis Title: 'Making Diagnosis Explicit'

What is good diagnostic practice? The answer is elusive for many medical students and equally puzzling for those trying to build effective medical decision support systems. Much of the problem lies in the difficult of 'getting at' diagnosis. Expert diagnosticians find it difficult to introspect on their own strategies, thus making it difficult to pass on their expertise.

Traditional knowledge acquisition methods are designed for gathering static domain knowledge and are inappropriate for the acquisition of knowledge about the diagnostic 'task'. More advanced knowledge acquisition methodologies, particularly those which focus on the modelling of problem-solving knowledge seem to hold more promise, but are not sufficiently practicable to allow anyone other than a knowledge engineer to operate directly. Given the difficulty experts have in accessing their own diagnostic strategies what is needed is a tool which would enable diagnosticians themselves to directly formulate and experiment with their own methods of diagnosis.

This research describes the development of a knowledge acquisition methodology geared specifically towards the exposition of medical diagnosis. The methodology is implemented as a toolkit enabling exploration and construction of medical diagnostic models and production of model-based medical diagnostic support systems. The toolkit allows someone skilled in diagnosis to articulate their diagnostic strategy so that it can be used by those with less experience.

Acknowledgements

First of all I would like to thank my supervisors, Peter Ross and Dave Robertson who have provided good-humoured advice, criticism and encouragement throughout.

The Artificial Intelligence Applications Institute have been very generous in allowing me to use their software, hardware and clients! In particular John Fraser, Ian Harrison and Ian Filby have been particularly helpful.

My evaluators Gill, Gordon, Ian, John, Kate, Nils, Rodney and Sharon were very generous with their time, and in sharing their opinions and experiences.

I would also like to thank the Kerr-Fry Bequest who supported the project 'Artificial Intelligence for Medicine in Developing Countries' which led to this work being undertaken, and the Imperial Cancer Research Fund who allowed me access to the knowledge bases of their 'Oxford System of Medicine' project.

This work was supported by studentship number 9031134X from the Science and Engineering Research Council.

Finally, Thanks to Howard, who believed in it all from the beginning, and Zoë, who arrived instead of Chapter 2 and is much more entertaining.

Contents

List of Figures

Chapter 1

Making diagnosis explicit

1.1 What this project is about

Medical diagnosis is a complex skill which, although it has numerous competent practitioners, remains something of a mystery to those who have to learn about it, either to become practitioners themselves or to build diagnostic support systems. Explicit strategies would be useful for both medical students and knowledge engineers but such strategies are often difficult to get hold of. In this chapter we discuss the role of explicit diagnostic strategies in both the teaching of medicine and in the building of effective, appropriate diagnostic support systems and examine some of the possible sources for such strategies. We then outline the aims and stages of a project to design, construct and evaluate a medical knowledge acquisition tool targeted at exposing the diagnostic task for these two application areas.

1.1.1 Explicit strategies for learners

> "...teaching methods have usually been oriented towards helping students in the learning of factual knowledge rather than providing feedback on their reasoning processes" ([Alpay 90] p. 274)

Those going through medical education often complain that diagnosis per se is simply not taught, or taught very crudely at a rudimentary level. Modern medical training is of necessity heavy on the factual side as medical practitioners in any speciality must have a

good grounding in general medicine and so much of the early years of a medical education is understandably given over to amassing large amounts of such 'raw' knowledge. This factual overload means that students have little time for structuring their knowledge, and can result in neglect of the teaching of diagnostic skills to such an extent that they must be invented from scratch by each neophyte. When students are finally exposed to the clinical diagnostic situation with real patients, after an extended period of gathering factual knowledge, they are taught to proceed using routine protocols. They are rarely asked to reflect on their reasoning processes or to consciously follow one in interacting with patients.

On the other hand, there is also a tendency on the part of experienced practitioners to avoid analysis of their ability, often referring diagnosis as an 'art', possibly a black one, thereby suggesting that it is unanalysable.

> "...clinical teaching is pervaded by the mystique of medical practice as an art, an art acquired by osmosis at the foot of the master. Too little attention is paid to the quality and methods of clinical teaching and few attempts are made to discern and enunciate clinical methods and strategies that make some physicians expert clinicians." (John T. Bruer, forward to [Evans & Patel 89])

Not only do students have little chance to reflect on their own diagnostic strategies, but they are not encouraged to do so by seeing their mentors engaged in such an activity.

There is an evident need to teach people diagnostic skills in order to produce effective diagnosticians but the combination of a requirement on students to learn vast amounts of factual information and the cageiness of expert practitioners about the characteristics of their diagnostic processes makes this very difficult.

1.1.2 Explicit strategies for diagnostic systems

There are many reasons for the failure or non-acceptance of medical (and other) decision support systems to be used in earnest in a practical context. One of the most damning faults for a knowledge-based system is when the system fails to please because the knowl-

edge it embodies is inaccurate or irrelevant to the application area and the representation is too rigid to allow easy alteration. For example, [Forster 90] gives an account of the problems with field trials of an archetypal primitive rule-based system based on a well known set of clinical algorithms (paper-based diagnostic flowcharts) used in developing countries ([Essex 80]). We can characterise such failures as emanating from

- problems with the content of the domain data,
- problems with the structuring of domain data,
- problems with inference procedures resulting in
- problems with apparent diagnostic behaviour,

and an inability to recover from these problems due to rigidity or lack of maintenance facilities. Incorporating an explicit diagnostic strategy into medical decision support systems can go some way towards addressing these problems where the inference strategy itself and thus the diagnostic behaviour of the system is inadequate. Where the diagnostic method proves to be unacceptable because it is either erroneous or unfamiliar, it can be very difficult and expensive to re-engineer if the method is too closely coupled with specific domain and case data. If the diagnostic procedure is *recognisable*, faults are more easily traced and a *separate* diagnostic component is much easier and less expensive to change. This also ensures that the integrity of unrelated parts, such as domain structures, can be preserved.

Existing medical AI systems also tend to incorporate only one method of diagnosis, if they have one at all. This means that the system is only suitable for use where its endusers are familiar with, or at least find acceptable, that particular diagnostic method. But people do diagnosis in different ways, both in different parts of the world and at different levels of specialisation.

> "...the diagnostic process, viewed as a monolithic structure, does not exist. Each clinician has his own pathway to diagnosis...Not only does the diagnostic pathway vary from clinician to clinician, but from patient to patient." ([Leaper *et al.* 73])

If the inference strategy is made explicit and discrete it is much easier to replace or customise for different users and contexts.

It is preferable that diagnostic systems are seen to be 'doing diagnosis' rather than simply running through data in a purely computational way. The method by which the system manipulates data in order to perform its assigned task, such as diagnosis, is its *control* mechanism. If a decision support system relies on a purely computational control mechanism such as simple backward chaining of rules, this can result in serious problems for explanation when things go wrong as well as with its general apparent behaviour. Unless a diagnostic system 'does diagnosis' in a recognisable manner it ceases to be a diagnostic *support* system and becomes more like a black box or 'guru' [Machanik 88]. Such systems are likely to be regarded as less reliable, more dangerous and their judgements less likely to be considered sound as the reasoning process does not correspond to that of a reliable expert. A system which performs diagnosis in the approved manner is likely to be seen as more 'trustworthy' as its reasoning processes, being based on reliable experts' behaviour, conform to a consistent known pattern.

1.1.3 Sources for diagnostic strategies

Given that diagnostic strategies are desirable both for pedagogic purposes and in building better diagnostic support systems, where can they be found? Getting diagnostic strategies from medical experts is notoriously difficult. Part of the reason for this is that these complex skills become 'compiled knowledge' where the practitioners may be very good at performing the task but have no insight into how they actually do it, the analogy being that the process is 'compiled' and they no longer have access to the original 'code' which initially outlined their strategy, for inspection.

> "With experience, thinking often becomes stereotyped and it may be difficult to say anything beyond 'I have had my solutions for a long time, but I do not yet know how I am to arrive at them' ([Hammond 80]). This should not, however, be an acceptable form of teaching. We should be able to analyse our processes, and students should expect explanations based on the scientific

method." ([Balla 85] p. 124)

We might expect that if the old hands no longer have insight into the structure of the skills they possess, we should be able to access the skills at a more formative stage. However, as we revealed above, diagnosis is often not taught explicitly as a skill to medical students, probably because of the problem just outlined, so it is difficult to pick up from training material or medical educators. Nor are students generally encouraged to reflect upon the development of their own strategies.

> "Although reasoning processes are discussed to a certain extent in medical apprenticeships, the structure of these processes is not explicitly presented to the students." ([Rodolitz & Clancey 89])

The medical education literature certainly does not abound with diagnostic models, explicit or otherwise. One of our evaluators (see Chapter 5), who tries to teach diagnosis, complained that there are plenty of books with 'diagnosis' written on them but nothing that can actually direct students in how to do it. For example, the many textbooks including words like 'Differential Diagnosis' in their titles contain nothing on diagnostic *strategy* whatsoever. These are detailed and unique descriptions of items of *factual* knowledge about diseases.

Students are sometimes introduced to the rudiments of decision analysis or Bayesian probability theory (e.g. [Balla 85] or [Sox 88]) but such numerical models give little or no support to a structuring of the medical reasoning process. They are purely computational methods which have no obvious parallel with the methods of human reasoning and systems which incorporate these methods are often criticised for not obviously 'doing diagnosis' (see section 1.1.2 above).

> "The principal group of restrictions inherent in numerical decision theories, is that they make no provision for reasoning about the decision process itself. Classical procedures cannot reflect on what the decision is, what the options are, what methods may be used in making a decision, what knowledge may be relevant, and so forth." ([Fox & Krause 91]

Another possible source is in specifications of existing software designed to perform the diagnostic task. However, diagnostic algorithms are often inadequately described or non-existent in specifications of many medical expert systems or decision support software. This is especially true of rule-based systems, like MYCIN ([Shortliffe 76]) which may rely mainly on simple backward-chaining as a control strategy, or strategy is buried within the rule base rather than being explicitly presented. In other cases the algorithm is not open to inspection because the system has been designed exclusively for one application. Where the control strategy is very closely linked with domain data a diagnostic algorithm becomes impossible to extricate. Much existing medical AI software therefore either has too simple a diagnostic strategy, none at all or the method cannot be disentangled from the specific application.

1.1.4 Knowledge acquisition for medicine

The process of transferring knowledge from an expert or other source through a knowledge engineer into a decision support system is a major part of the effort in building such systems and this 'knowledge acquisition bottleneck' is particularly acute in medicine.

> "There is little doubt that the process of mapping the ill-structured knowledge of a medical subspeciality into a form suitable for machine encoding is amongst the most difficult and time-consuming parts of the process of building a KBS." ([Lanzola & Stefanelli 91])

Often it involves both the knowledge engineer becoming an expert, in a limited sense, and the expert becoming something of a knowledge engineer. One of the more promising sources of diagnostic strategies, then, is in the literature and software surrounding knowledge acquisition for diagnosis. There are two potential sources; those techniques which assist in the production of diagnostic strategies by, or extraction of such strategies from the experts themselves and those methods which incorporate their own diagnostic strategies.

Knowledge acquisition methods and tools that are usable by domain experts, however, generally do not allow the construction of an explicit diagnostic strategy, indeed they

rarely address the issue of a control mechanism at all. Such methods tend to be geared only towards the acquisition of static domain data and the resultant system is likely to use a very simple 'computational' algorithm.

There are some diagnostic algorithms described in the literature surrounding *modelling* methods (e.g. Clancey's heuristic classification [Clancey 85]) and it is these methods which emphasise the control aspect of decision support systems. Modelling methods of knowledge acquisition are more fully described in Chapter 2. However there are not many practical implementations of the current modelling paradigms as knowledge acquisition *tools* and those tools which exist generally do not have any facilities for explicit strategy modelling; models of diagnostic strategy come pre-assembled and can be customised. These methods are also not generally directly usable by experts in the areas of application without the help of a knowledge engineer. Further, as such methods tend to cover multiple tasks of which diagnosis is only one, their diagnostic strategy is often presented as *the* diagnostic method. This is fine if it also corresponds to the method used by, and recognised by, both the expert and the enduser, but, as we have seen, there are many ways of doing diagnosis. Knowledge acquisition tools would seem to provide a promising opportunity to access diagnostic methods, both for medical education and for the building of better diagnostic support systems. However, at present what is missing is a method which allows the specification of multiple diagnostic strategies by learners, teachers and experts as well as knowledge engineers.

A usable knowledge acquisition methodology for analysing diagnosis can have several roles to play in the educational situation. It can both encourage and assist experts in exposing their own diagnostic algorithms to pass them on or even improve them. It can be used to find out the characteristics of good diagnostic practice as exemplars for learners, describing the behaviour of the good practitioner in terms that are both understandable and replicable. If the method allows synthesis as well as analysis students can also benefit from being able to synthesise their own diagnostic strategies to see how they behave in a diagnostic situation. An effective analysis method is also a necessary step towards avoiding many of the pitfalls of existing diagnostic support systems (see section 1.1.2) and building more effective, appropriate systems. It can be used, as in the educational

situation, to develop an adequate representation of good diagnostic practice in order to eventually replicate or emulate that practice in diagnostic support systems.

This thesis describes the development of a knowledge acquisition methodology geared specifically towards medical diagnosis and its accompanying prototype toolkit designed to enable the construction of medical diagnostic models and the production, by direct manipulation, of model-based medical diagnostic support systems. The methodology particularly encourages the analysis of diagnosis as a *task*, emphasising

- the separation of domain and task knowledge in medicine
- the representation of domain and task knowledge in models
- the construction of models of diagnosis from atomic subtasks
- the utility of flexible diagnostic strategies which are responsive to domain and case context

We have implemented a toolkit which is geared specifically towards medical diagnosis and provides facilities both for domain and task modelling. The toolkit implements this methodology with

- individual tools for building separate task and domain models
- a library of diagnostic subtasks
- the use of constraints as a mechanism to control the responsive behaviour of models
- a critiquing mechanism for refining the behaviour of models.

1.2 Aims

The aims of this research are primarily to develop a realistic method of making diagnostic strategy explicit and to produce a knowledge acquisition tool implementing that method which allows the analysis and synthesis of diagnostic strategies so that learners can explore such strategies within a real framework and domain experts can create and

customise them to be incorporated flexibly into diagnostic support systems. We aim to do this by

1. developing a methodology for analysing diagnosis;
2. designing a tool which would implement this methodology enabling
   (a) the exploration and exposure of the diagnostic task,
   (b) the construction of diagnostic task models,
   (c) the construction of domain models,
   (d) the building of better diagnostic support systems which can use these models,
   (e) the building of complete diagnostic support systems by the experts themselves,
3. implementing the tool;
4. testing the tool.

1.3 Structure of thesis

We explore the appropriateness of existing methods and tools for knowledge acquisition to medical diagnosis, concentrating particularly on modelling methods and tools, and attempts to model the diagnostic process. We then outline and motivate our own methodology. We describe the purpose and function of the tool which implements that methodology. We describe the method and results of an evaluation of the methodology and tool. We conclude with an analysis of our achievements, limitations on the existing method and tool, and explore the potential for improving the tool and for applying the method to other tasks and domains.

Chapter 2:

Tools and Methods for Knowledge Acquisition

We explore rule induction systems, psychological methods, template tools, knowledge acquisition tools specific to medicine and modelling tools, looking particularly at task

modelling in medicine and the current climate of task modelling and the lack of suitable, usable tools for building diagnostic support systems. We suggest what system characteristics would fill this gap.

Chapter 3:

A Methodology to Analyse Diagnosis

The methodology itself is broken down into the separation of domain and task knowledge, the modelling of domain and task knowledge, modelling with subtasks and the construction of responsive task models. Each of these features is explored in detail and described in terms of its application areas (why it is useful) and how it influences the design of a practical system.

Chapter 4:

TOMKAT: A Tool to Implement the Methodology

We explain how the methodology of Chapter 3 guides design decisions for the implementation of a model-based knowledge acquisition tool. We describe the structure and functioning of TOMKAT (Task Oriented Medical Knowledge Acquisition Toolkit), a system which allows medical experts to build their own diagnostic support systems by providing facilities for the construction of a domain model and a task model.

Chapter 5:

Evaluation

We describe the aims, methods and results of a practical evaluation of TOMKAT conducted with individuals of varying levels of medical and AI expertise. The study concentrated on *concept* evaluation, where the ideas expounded in Chapter 3 were tested out on the evaluators and *product* evaluation where the effectiveness of the implementation of those ideas in the TOMKAT system was tested. The evaluation was conducted using structured interview and structured task techniques. As the toolkit is a laboratory prototype only, this evaluation was regarded as *formative* in that evaluators were encouraged

to suggest alterations and improvements to design, as well as being *summative* of the existing product's achievements.

Chapter 6:

Conclusions

We assess how far the initial aims have been met, examining successes and failures in the methodology and software uncovered by the evaluation. We discuss the extensibility of the work in terms of turning the prototype into a real system and applying the methodology to tasks other than diagnosis, in particular, to scheduling.

Appendices:

A: We describe implementation structure and file storage of the TOMKAT system.
B: Evaluation documentation includes the concept outline, product outline, and structured interview.
C: A user manual details how to go about building diagnostic systems with TOMKAT, and shows domain and task models under construction.

1.4 Summary

If diagnostic strategies were readily available there would be little excuse for their not being used in the teaching of medicine and incorporated into decision support system design. However, they are not, and we hope to be able to 'get at' diagnosis from many possible sources. The method must have the potential for exposing, representing and constructing diagnostic algorithms, to allow experts access to their own strategies, for teaching purposes and to build effective diagnostic support systems. As a knowledge acquisition method it must be capable of acquiring task knowledge over and above domain knowledge and permit this to be represented adequately. It should provide both a means to decompose the good practitioner's art and a representation which will be capable of reflecting that behaviour so that it can be replicated.

Chapter 2

Methods and tools for knowledge acquisition

In the previous chapter we saw how medical experts, teaching materials and the algorithms of diagnostic support software do not provide rich pickings as sources of diagnostic strategies. In this chapter we investigate the potential of various types of knowledge acquisition methods and the tools implementing these methods for exposing diagnostic strategies and building medical diagnostic support systems. We look in particular at *modelling* methods which seem to be the most promising for our purposes, examining several systems which do something very like task modelling in the domain of medicine, although the systems we have uncovered are by no means exclusively knowledge acquisition systems. Finally we outline the requirements of a knowledge acquisition system incorporating task modelling for medicine.

This is not a review of *all* knowledge acquisition techniques and tools, merely a representative sample of types which might be of use in the context of knowledge acquisition for medicine.

2.1 Non-modelling methods

Non-modelling methods are what might be called the 'traditional' knowledge acquisition methods, often geared towards building rule-based systems. Acquiring knowledge for rule-based systems is a non-trivial exercise as people do not generally think in production

rules and forcing experts to express their knowledge directly into such an alien formalism involves an unnecessary effort on their part to restructure their thinking around the technology. Rule induction systems are an attempt to assist the rule finding and making process by simply asking the user for objects, attributes and values over which rules can be induced. 'Psychological' methods provide more assistance, again in producing rules, by prompting for differential properties amongst groups of objects. 'Template Tools', specifically for diagnosis/troubleshooting, provide a framework into which experts can fit the details of their particular diagnostic/troubleshooting domain.

2.1.1 Rule induction

Much current work using the inductive method and tools which implement it are based on the original work of [Quinlan 79]. Inductive inference is fundamentally the making of a general statement on the basis of a set of individual case instances. The justification of the use of rule induction as a knowledge acquisition method is the observation that whereas experts may be very capable of supplying case examples of their decisions, they are not often competent at expressing the sort of rules that can be inserted directly into a knowledge base. The inductive method may even be used without the presence of an expert where, for example, a large amount of case data are available. A rule induction tool is supplied with a 'training set', a lot of domain examples of expert classification consisting of sets of attributes and values and the classifications assigned to them. Rules are then induced upon the most efficient set of attribute value questions which lead to the classification, giving a decision tree which enables the categorisation of any object in the training set to be determined from the values of its attributes. Induction is therefore an appropriate method for domains where simple classification is all that is required. Rule induction techniques of various flavours, have been used quite widely in medicine with some degree of success ([Funk *et al.* 87], [Pirnat *et al.* 89], [Schijven *et al.* 89]). A similar method ([Pankhurst 79]) was used to produce a well known set of clinical algorithms (paper-based diagnostic flowcharts) used in developing countries ([Essex 80]).

Tools which use the inductive method are either rule-based shells with the addition of an inductive component (Crystal, KnowledgeMaker, XiRules) or tools which are basically an

induction package with the additional facility to run the induced rules (e.g. Expert-Ease, described in detail in [Bloomfield 86]).

2.1.2 Psychological methods

A major difficulty in the use of the inductive method is the selection of relevant objects and attributes over which to induce the rules. Assistance with the problem of selecting those crucial objects, attributes and values with which to make rules is on hand with another family of tools; those based on the 'psychological' method of Personal Construct Theory as described in [Kelly 55]. The technique known as Repertory Grids involves prompting the user for objects in the domain and then eliciting differential comparisons amongst randomly selected object triples in order to tease out the appropriate attributes and values. The expert suggests an attribute common to two of the objects but not shared by the third and all other objects are ranked by them on that attribute or 'construct'. This is repeated until the supply of constructs dries up, or boredom sets in. The grid consists of a representation of all objects ranked on all constructs.

There are tools which combine these techniques of rule induction and Repertory Grids (e.g. Neuron Data's Nextra, Intelligence Ware's Auto-Intelligence [Parsaye 88]). These operate by prompting the user for objects attributes and values, then inducing rules over the information so supplied.

2.1.3 'Template Tools'

One other type of system which should be mentioned is what are sometimes called model-building tools ([Aylett 90] provides an incisive summary of their capabilities). I have chosen to call these Template Tools to avoid confusion with true modelling tools described later in this chapter. These are not the same as the modelling tools as they simply provide templates for one type of datastructure and only allow one method of inference. There is no identifiable knowledge acquisition methodology associated with these tools and they are very application specific. The tools supply primitives like 'test' and 'result' out of which the decision task can be built. They are primarily for building

the protocols and flowcharts which form the basis of fault-tracing or 'troubleshooting' systems. They could have potential application for a simple medical system but this has not so far been attempted. Examples of tools in this category are Intellicorp's KLUE ([Karel & Kenner 89]) and Carnegie's TestBench, both adjuncts to larger knowledge engineering toolkits (KEE and KnowledgeCraft respectively).

2.1.4 Problems with non-modelling methods

Rule induction has many problems as a knowledge acquisition tool. It will not necessarily produce rules which are equivalent to those used by the expert, and may in fact result in particularly implausible lines of questioning. The expert may 'know' that a particular attribute is very important in decision making whilst another is not, but there is no way of expressing this and the inductive method, being essentially blind, may light upon the unimportant attribute simply because of idiosyncrasies of the training set. It is perfectly possible to select attributes that are classificatorily efficient, in that every individual has a different value, but diagnostically meaningless (such as age). As [Bramer 87] points out we need to assess

> "...not whether a set of rules can be induced, but how to ensure that the induced rules capture the underlying causality of the domain and hence have a high level of predictive power." ([Bramer 87] p. 15)

The question of assessing the induced rules raises further problems. If the expert has to assess the 'bare' rules they will be in a similar position to the expert who has to enter their knowledge directly in rules; it is not a familiar format. The other main difficulty is finding the appropriate entities, attributes and values in the first place, from which the rules will be generated. The overhead for the expert to learn how to use induction as a knowledge acquisition tool accurately is thus high, and the resultant systems may not be plausible enough to justify the effort. Induction is a good method for situations where what is required is a fairly rigid system to solve an essentially classificatory problem and where the domain has little inherent structure. It is not so appropriate for areas where theoretical knowledge is equally or more important. Although medical systems

have been built using rule induction, as mentioned above, it is difficult to see how such systems could embody a chosen diagnostic method let alone allow that method to be made explicit.

The Repertory Grid technique, although helpful in exposing differential qualities in the domain, does not provide enough domain structuring for experts to use alone successfully. The tools may not require much learning investment, but experts also find them spectacularly boring to use, precisely because of the lack of domain structuring. Filling out low level differential descriptions of objects can be immensely tedious and unrewarding. It should also be remembered that these systems are still intended to produce rules, and the inferential structure of the resultant system will always be the decision tree. Although such systems could potentially produce a fully working system from start to finish of the knowledge acquisition process, their makers generally recommend that the output be used simply as a first stab input to a more sophisticated knowledge engineering tool, again requiring a knowledge engineer for the final twiddling e.g. Nextra's output is intended as input for Neuron Data's knowledge engineering tool Nexpert Object. The psychological methods are again appropriate for acquiring factual knowledge, such as the discriminating characteristics of diseases, rather than information about diagnostic methods.

Template Tools are very task-specific, being aimed exclusively at the diagnostic or troubleshooting task, and it is really necessary that the experts already have their knowledge mapped out in some sort of decision graph manner, already knowing what the entities attributes and values are, and having some insight into the structure of the task. They do not really do much knowledge acquisition at all, let alone allow the exposition of diagnostic strategies. The problem with these tools is that although there is some knowledge structuring, they do not supply much else.

Any method which is geared towards producing rules must face the many criticisms which can be leveled against rule-based systems. There are general difficulties in change, maintenance and update. Rule-based systems are not easy to debug and extend because rules may be very inter-dependent. As mentioned previously, rules are often an alien representation of knowledge for many people. This means that an assessment of the

output of a system that produces rules can be difficult for the expert. Even if the expert is familiar with rules, systems like this can easily produce unfamiliar rules or the ordering of questions in the product can seem bizarre. Rule tracing as an explanation method is at best awkward and since the system has no knowledge of itself it cannot explain why its rules work. Most seriously from the point of view of exposing the diagnostic method, if all the knowledge is in the same format, the rule, it is difficult to separate control from other sorts of knowledge, and if there is no explicit, separate process, there is no possibility for the user to follow the inference procedure, either to concur or intervene. Rule-based systems tend to rely on a simple control strategy such as backward chaining, and the issue of how a system behaves as regards a particular task, like diagnosis, is not addressed. In fact it is difficult to see how one would acquire this type of knowledge with one of these methods, or what it would mean in such a context. The 'diagnostic method' would be something like backward chaining. Such a diagnostic strategy would be of little use for our purposes not being based on anything real diagnosticians do and having little or no pedagogic utility.

Generally, these methods can be very useful for structuring facts and relationships between them but are not adequate for acquiring process information of any but the simplest kind.

2.2 Modelling methods

The main problems with the non-modelling methods then are that although they are very good for acquiring factual domain knowledge they are not appropriate for acquiring knowledge about how diagnosis proceeds. They are geared towards producing 'first generation' rule-based systems which tend to have a simple and invariate control strategy and shallow model of expertise which has little value for teaching purposes and does not mirror the diagnostic strategies used by experts.

Turning expert knowledge into rules, which are a convenient formalism for computational manipulation but can be alien to the expert whose knowledge is being garnered, is no longer a universally popular strategy. What has taken the place of rule-gathering for

the knowledge engineer is the process of *modelling*. Some proponents of the modelling approach would claim that the building of models is now just about *all* of knowledge engineering.

> "...today knowledge engineering is approached as a modelling activity: the heart of the work of the knowledge engineer lies in the actual construction of models." [Wielinga *et al.* 92b] p. 4

There are currently two main schools of modelling as a knowledge acquisition method; the KADS school in Europe ([Breuker & Wielinga 87]) and the Generic Tasks school in the USA ([Chandrasekaran 86]). They are fundamentally very similar, although they seem to have come about by parallel evolution. Model based approaches hypothesise that it is useful to view knowledge as being of different types. Specifically, the distinction between content knowledge and problem solving knowledge is of value, although more, and finer distinctions may be made amongst knowledge types. Model based approaches to knowledge acquisition assume that a high level description or abstract representation (the model) of problem solving is of more value than an ad hoc application dependent one. It is evident that there are several different types of diagnosis and that different authors have different ideas of the algorithm(s) involved. What is not at issue is that the high level task of diagnosis can be broken down into more manageable subparts or subtasks, although the exact nature of these individual building blocks may vary. Most modelling tools and methods attempt to provide an environment for building systems geared towards a wide variety of problem solving tasks. Both KADS and Generic Tasks describe tasks other than diagnosis, but the diagnostic task is the one which has been the target of much research. Although we are primarily concerned with modelling the task of diagnosis, this should be seen simply as a task amongst other tasks.

2.2.1 Generic Tasks

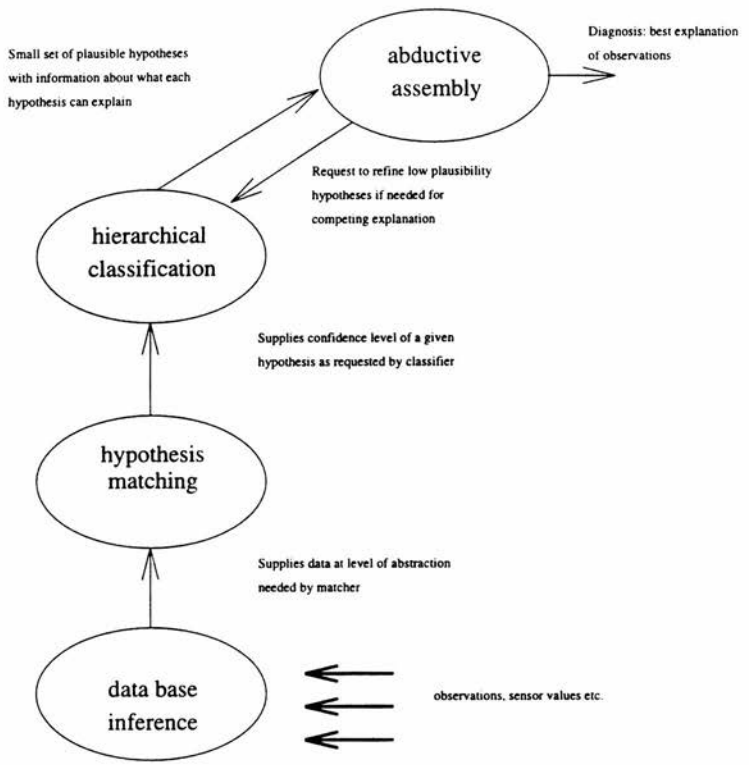
The Generic Tasks approach sees its aims as being to find the primitive types of knowledge based problem solving and principled ways to build complex problem solving out of these primitives. Generic tasks are the building blocks or reasoning primitives which

can be used for constructing high level compound tasks. They are characterised by their input and output, the control regime they implement and the knowledge constructs they use. The control regime describes how the generic task can be further subdivided into subtasks, e.g. hierarchical classification consists of establish, refine, reject and prune but it is unclear whether these subtasks are also to be regarded as generic and reusable. Generic tasks described in the literature are hierarchical classification, hypothesis matching, knowledge directed information passing, abductive assembly, hierarchical design by plan selection and refinement and state abstraction.

Diagnosis itself is not regarded as a generic task but a compound task which is decomposable into generic task subparts and can be implemented in more ways than one. An example given of a model or generic task architecture for diagnosis, in this case 'diagnosis with compiled knowledge', is built out of four generic tasks, data abstraction and inference, hypothesis matching, hierarchical classification and abductive assembly. In this, essentially fixed architecture, the structuring information (the algorithm) for 'diagnosis with compiled knowledge' can be diagrammatically represented as in figure 2.1 ([Chandrasekaran 88] p. 190).

The original idea was that there should be a tool for each generic task, resulting in a generic task 'Toolset'. Tools were developed, e.g. CSRL (Conceptual Structures Representation Language) for hierarchical classification, HYPER (HYpothesis matchER) for hypothesis matching. As yet there are no commercial tools which implement the Generic Tasks approach.

> "With the identification of generic tasks languages can be developed that encode both the problem solving strategy and the knowledge that is appropriate for solving problems of that type. These languages facilitate the development by giving the knowledge engineer access to tools which work closer to the level of the problem, not on the level of implementation languages such as rules or frames."([Karbach *et al.* 90])



(Adapted from Chandrasekaran 1988)

Figure 2.1: Generic Tasks' model of diagnosis

2.2.2 KADS

The KADS methodology encompasses the whole life cycle of application design using a model based approach. The modelling effort which leads to, in effect, a system description is known as *expertise* modelling. Expertise modelling involves acquiring three types of knowledge; domain, inference and task. Domain level knowledge is static, factual knowledge about objects, properties and relations. Inference level knowledge identifies which inferences can be made, 'knowledge sources', and how domain level elements are grouped for problem solving, 'meta classes' (I will continue to refer to 'knowledge sources' and 'meta classes', although the latest version of KADS, KADSII names them differently). Task level knowledge identifies the structure of the reasoning process.

If we examine the potential for exposing diagnostic strategies or constructing such strategies for incorporation into a diagnostic support system, we find that the KADS methodology emphasises model *refinement* rather than model *building*, the system builder being encouraged to choose one of a selection of interpretation models from a library, as the following definition makes clear:

> '**Model Library:** A library of *generic models* that is maintained within the KADS methodology. KADS is model-driven and the *initial model* chosen to drive the *internal stream* of analysis is often chosen from this library rather than being built from primitives.' ([Hickman *et al.* 89] p. 179)

The interpretation model library supports a variety of known tasks arranged in a hierarchy of analysis tasks like classification and diagnosis, modification tasks like repair and maintenance, and synthesis tasks like planning and design. This choice of interpretation model is regarded as the initial objective of building the expertise model. Once the interpretation model has been chosen it is instantiated and altered to fit the domain and is known as the *inference structure*. The methodology does theoretically allow the construction of a model from primitives if none of those in the KADS interpretation model library are appropriate, although this method is not emphasised. The knowledge sources are of a finer granularity than the Generic Tasks' primitive elements. A knowledge source is a primitive inference function whose inputs and outputs (meta-classes)

describe the roles that domain primitives play in problem solving. In fact they are more like the parts of which Generic Tasks themselves are made. This means that not only are existing known tasks (like diagnosis, modelling, planning) catered for, but there is also the potential to build new tasks out of construction elements.

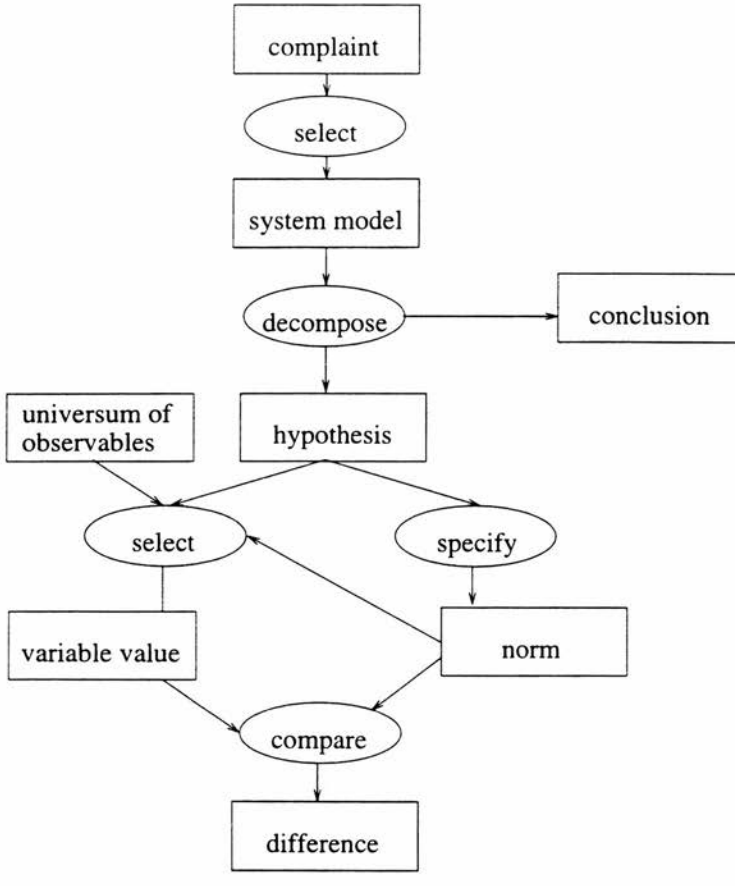
The KADS literature also mentions several different task models of diagnosis; single-fault diagnosis: heuristic classification, systematic diagnosis by causal tracing or by localisation and multiple fault diagnosis. The algorithm for systematic diagnosis is represented diagrammatically as in figure 2.2. Note that this algorithm, as well as including the primitive subtasks (knowledge sources) also includes the domain elements (meta-classes) which are required as inputs and outputs. To use the KADS methodology for building diagnostic support systems it would be necessary to first select an appropriate interpretation model from the several diagnosis models in the library and then customise it to correspond to that of the expert.

The main current software implementation of the KADS methodology is KADSTOOL, produced by ILOG. This is a modelling tool dedicated to expertise modelling and incorporating tools for modelling domain, inference and task knowledge.

2.2.3 Problems with modelling methods

The Generic Tasks approach seems almost suitable for our purposes. Problem solving and domain knowledge are separated and an abstract model is built from subparts so that there are several possible models of diagnosis. Unfortunately there are no commercially available tools implementing the methodology and it is still primarily for use by the knowledge engineer rather than the expert. The primitives out of which the task models are built are not likely to be readily identifiable by medical domain experts. There is also no information on how domain knowledge is to be acquired or incorporated into systems. The potential for exposing diagnostic strategies or constructing them is thus much better than with the non-modelling methods but not quite perfect.

KADS has the advantage over Generic Tasks of incorporating domain knowledge and supporting the building of a domain model. It also now has a commercially available



(Adapted from Hickman 1989)

Figure 2.2: KADS' model of diagnosis

tool implementing the methodology. However the KADS methodology forces a very high learning overhead. It requires the services of a knowledge engineer for implementation and is not an ideal environment either for domain experts to create and customise their own diagnostic strategies, and even less so for domain learners to explore diagnostic strategies. This is probably because the KADS methodology has a very broad and ambitious scope. In covering the full spectrum of task types, from synthesis tasks like design to analysis tasks like diagnosis, the methodology seeks to provide general problem solving facilities. In this situation the specificity which is necessary for the domain expert is lost. KADS has much appeal for the knowledge engineer but not for the diagnostician. The KADS methodology also emphasises model refinement; the tailoring of a ready made interpretation model to suit a specific purpose, rather than model building, and there is one base model of diagnosis.

Generally, modelling methods go some of the way towards providing what we want. The separation of knowledge levels enables control or problem solving knowledge to be extracted and examined independent of domain application details. The building of models of task knowledge allows diagnostic strategies to be made explicit. The division of high level tasks into more primitive sub-parts means that more than one model of diagnosis can be built using the same atomic elements. However the implementation of the modelling methods is not ideal for our purposes. Because the methods are very generic, almost to the extent of being general problem solving methods, tools to implement them can be prohibitively large and unwieldy. Successful use of tools like these depends on the presence of the knowledge engineer and understanding the technique requires a great deal of background AI knowledge. They can potentially deal with a multitude of problem solving tasks but the learning overhead is very high. Perhaps because of their wide general problem solving scope these methods tend to be limited in the variety of diagnostic models they can offer. There is nothing wrong with offering off-the-peg models which the user can choose as being relevant to their purpose but, as we have seen, people do perform diagnosis in different ways. If there is no model which corresponds to the user's own, they will be stuck.

2.3 Modelling applications: Task modelling in medicine

Generic Tasks and KADS then are not totally suitable for our purposes because they are too general, they do not allow experts to do it themselves and their models of diagnosis are not varied enough.

The systems explored in this section are all geared towards medical applications, and incorporate models of specifically medical diagnosis. Such models tend to be much more explicit and detailed than the models of general diagnosis previously mentioned, because these systems do not have the wide, but shallow, scope of the general modelling methods. All again use the idea of a diagnostic model built out of subtasks.

2.3.1 NEOMYCIN and GUIDON-MANAGE

NEOMYCIN and GUIDON-MANAGE are systems which grew out of the MYCIN and EMYCIN experiments ([Shortliffe 76]). Whereas MYCIN does not organise or use its knowledge in the way an expert does, because it is designed only to simulate expert performance and not the expert process, NEOMYCIN attempts to produce a psychological model of medical problem solving with explicitly separate representation of the causes and processes of disease and of strategic knowledge in a domain independent form. A series of educational systems which use NEOMYCIN's knowledge base were developed, of which GUIDON-MANAGE provides an environment for explicit manipulation of the diagnostic task. NEOMYCIN and GUIDON-MANAGE are described variously in [Clancey & Letsinger 84], [Clancey & Bock 88], [Clancey 88] and [Rodolitz & Clancey 89].

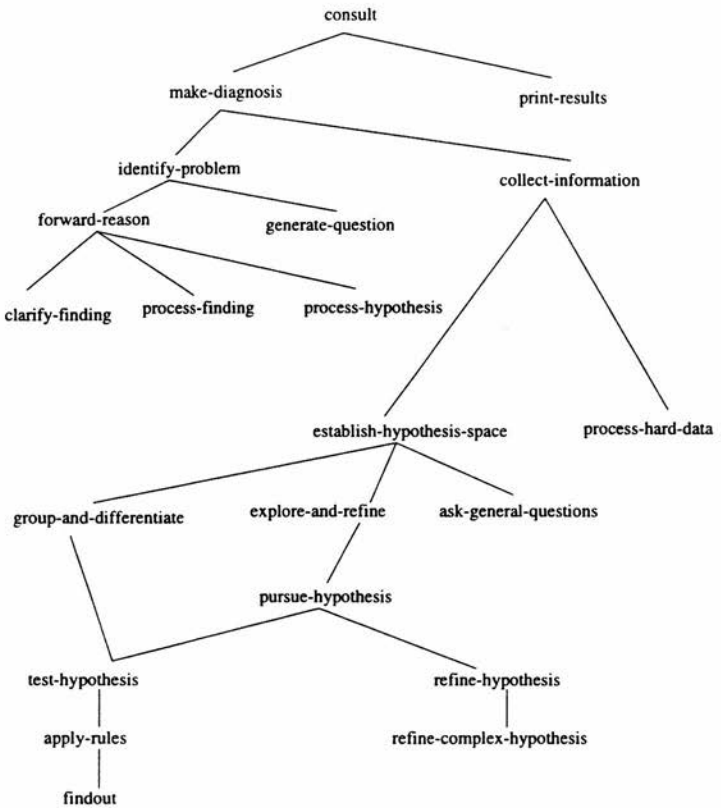
NEOMYCIN has one high level task model in which diagnosis is assumed to proceed in the following manner. There is a hierarchy of diagnostic subtasks, each being a descendent of the overall task of diagnosis (called 'consult'). There is also a domain hierarchy of diseases. After initially gathering a patient description and presenting symptoms, the problem solver at the beginning of the consultation usually lands in the middle of a hierarchy of diseases. They then look up the domain hierarchy of diseases to find evidence for a general class of disease and then look down to find the specific instantiation, i.e.

form a set of possibilities that might include the right answer, then narrow that down. This general model is represented by a *task hierarchy*, a part-of rather than a subclass hierarchy so that the subtasks of each task are fixed (e.g. the task forward-reason always consists of all and only the tasks clarify-finding, process-finding and process-hypothesis). Subtasks may be children of more than one higher task (e.g. test-hypothesis is a subtask of group-and-differentiate, pursue-hypothesis and findout) and there can be recursive loops. NEOMYCIN's diagnostic model is represented in figure 2.3.

Although NEOMYCIN has effectively only one diagnostic model it is not completely rigid. There are conditions on the execution of low level tasks which act to ensure the appropriate firing of subtasks according to the incoming case data. Each task has an associated group of *metarules*. A metarule has conditions on whether the task's subtasks can be tried, i.e. they determine which domain relations will be used (contrast hypotheses, focus on hypothesis, refine hypothesis, confirm hypothesis, findout whether finding is present). High level metarules exercise constraints on task ordering (e.g. history and physical always precedes labdata collection). Each task also has a description of how its metarules are to be applied (using one of try-all, try-all till one succeeds, try all iteratively till none succeeds, and each time one succeeds start again until none succeeds).

The tasks implemented in NEOMYCIN are based on the different reasons for asking a question: to test a hypothesis, to follow up a previous question, to discriminate between hypotheses, because a question has been automatically triggered, to determine the presence of a finding, to establish that a history is complete. The main elements of the task hierarchy are:

- **Consult:** invokes **make-diagnosis**.
- **Make-diagnosis:** invokes **identify-problem, review-differential** and **collect-information**.
- **Identify-problem:** asks for demographic information and the presenting symptom. Then invokes **forward-reason**. If the differential remains empty, invokes **generate-questions**.



(Adapted from Clancey, 1989)

Figure 2.3: NEOMYCIN's model of diagnosis

- **Forward-reason:** invokes **clarify-finding** for each finding, **process-finding** for each serious or non-specific finding and **process-hypothesis** for each hypothesis
- **Clarify-finding:** (finding) asks history-type questions about finding e.g. duration.
- **Process-finding:** (finding) antecedent (causal and definitional) rules, generalisation relations, trigger rules that suggest a hypothesis, consequent rules that use soft data, consequent rules that use hard data subject to constraints on economy of effort and efficiency.
- **Process-hypothesis:** (hypothesis) maintain the differential and do forward reasoning.
- **Findout:** (finding) How the problem-solver makes a conclusion about a finding they want to know about.
- **Applyrules:** How domain rules are applied.
- **Generate-questions:** to ask questions that will suggest some new hypotheses. Questions are generated from **ask-general-questions, elaborate-datum**, any rule that wasn't previously applied because it required new subgoals and asking 'any more information?'.
- **Ask-general-questions:** domain-specific history questions asked in a fixed order.
- **Collect-information:** does hypothesis directed reasoning.
- **Establish-hypothesis-space:** three ordered rules about ancestor hypotheses and general questions iterate
- **Group-and-differentiate:** attempts to establish categories that should be explored.
- **Test-hypothesis:** for directly confirming a hypothesis.
- **Explore-and-refine:** for choosing a focus hypothesis from the differential.
- **Pursue-hypothesis: test-hypothesis** then **refine-hypothesis**.

- **Refine-hypothesis:** puts children of the hypothesis in the differential. If there are more than four, **refine-complex-hypothesis**. For each child, **apply-evidence-rules**.
- **Refine-complex-hypothesis:** Select the common and unusual causes of the hypothesis.
- **Process-hard-data:** Find out what 'hard' findings (like labdata) support hypotheses on the differential. Return to **group-and-differentiate** and **explore-and-refine** new hypotheses as necessary.
- **Review-differential:** prints out the differential.
- **Apply-evidence-rules:** apply rules that support the hypothesis using given findings.

As compared with the KADS and Generic Tasks models of diagnosis, NEOMYCIN's model is much more detailed. This is probably because it is a general model of *medical* diagnosis rather than a model of *general diagnosis*. It is also closely coupled to domain structures.

GUIDON-MANAGE uses the underlying structure of NEOMYCIN, its tasks and metarules, to allow students to experiment with diagnostic strategy. Whereas NEOMYCIN has an essentially fixed diagnostic strategy (which was based on information from one expert informant), GUIDON-MANAGE asks the student user to choose which task to do next, within certain limits, creating an environment which allows the interactive, dynamic construction of a diagnosis.

> "NEOMYCIN uses a particular order in its problem solving; however this is not the only correct strategy. GUIDON-MANAGE must therefore be prepared to accept alternate strategies, within certain bounds." ([Rodolitz & Clancey 89] p. 316).

The aims are to introduce students to a language of problem solving and provide an

environment for learning and exploring this language through co-operative problem-solving with an 'expert'; NEOMYCIN itself.

The student does not have access to the complete hierarchy of NEOMYCIN's tasks. The top level ones (from consult to pursue-hypothesis) were deemed too abstract and the low level ones that manipulate domain rules were deemed too specific. The tool works by running NEOMYCIN's interpreter until it hits a task that lies in the student range. The model is then suspended and the student performs a diagnosis by selecting (mid level) tasks that the system will execute. The chosen task's metarules are then applied and its subtasks executed. These will result in activities like the asking of questions to gather case information. The high level tasks are only used to generate help messages about what tasks might be appropriate.

The tasks available to users of GUIDON-MANAGE are equivalent to the mid level tasks of the NEOMYCIN hierarchy.

- **Clarify** a finding: gathers more information about finding (take history of symptom).
- **Consider** a finding: looks for relationships between the given findings and categories of hypotheses or findings.
- **Ask about** a finding: asks whether it is present. = FINDOUT
- **Test** a disease: tries to rule disease in or out = TEST-HYPOTHESIS
- **Refine** a disease: finds the more specific examples of one disease.
- **Collect labdata for** a disease: requests labdata that would rule disease in or out.
- **Get results from** a labtest: asks for all results of a labtest.
- **Add to differential** a disease: puts hypothesis in differential.
- **Remove from differential** a disease: removes hypothesis from differential.

2.3.2 DEMEREST

Laurence Alpay's DEMEREST project ([Alpay 90]) aimed to model various stages of medical expertise from student to expert in order to build a system that can asses the competence of a student by identifying their use of various diagnostic strategies and thus determining their level of expertise. The DEMEREST system itself assesses the student's reasoning to assign a level of expertise and produces a plan corresponding to the application of these strategies. Models of diagnostic reasoning are built from these strategies which correspond to the primitive subtasks or atomic parts out of which other modelling methods compose their models of diagnosis. The use of subtasks again means that many different possible models of diagnosis can be constructed.

> "Students and more experienced physicians combine reasoning strategies in different ways and therefore do not necessarily use the same form of reasoning" ([Alpay 90] p. 125)

The strategies or subtasks implemented in DEMEREST are as follows:

- **Generalisation:** generate a general hypothesis from a specific one by establishing the parent categories of the hypothesis.
- **Specialisation:** generate a specific hypothesis from a general one by establishing the more specific examples of the hypothesis.
- **Confirmation:** validate a hypothesis by asking about findings.
- **Elimination:** rule out a hypothesis by asking about findings.
- **Problem refinement:** Gather more details on a finding.
- **Hypothesis generation:** generate a hypothesis from a finding.
- **Anatomy:**. Generate information using anatomical knowledge.

DEMEREST is intended eventually to be incorporated as the student modelling component of an intelligent medical tutor. The student model would be used to guide the

teaching module of the tutoring system. The 'developmental modeller' component contains models of clinical reasoning at different levels of expertise. Novices and experts all have access to the same subtasks, but it is how they combine them that varies. The differences between these models seem to be primarily of the nature of subtask *ordering*.

Although DEMEREST is not really in the business of *building* models of diagnosis it is included here because it incorporates many of the features which would be characteristic of a tool which would suit our purposes. It recognises the need to separate domain and problem solving knowledge. It incorporates models of problem solving knowledge and of domain knowledge. Task models are built out of subtasks at a mid-level of abstraction likely to be recognisable by domain experts. It recognises that there may be multiple models of clinical diagnosis dependent on both individual preference and level of expertise. It is also of interest in that it reinforces the choice of subtasks of other diagnostic models and modelling systems. All the 'strategies' are recognisable as similar to elements of NEOMYCIN, and the author fully acknowledges the similarity to this and other systems.

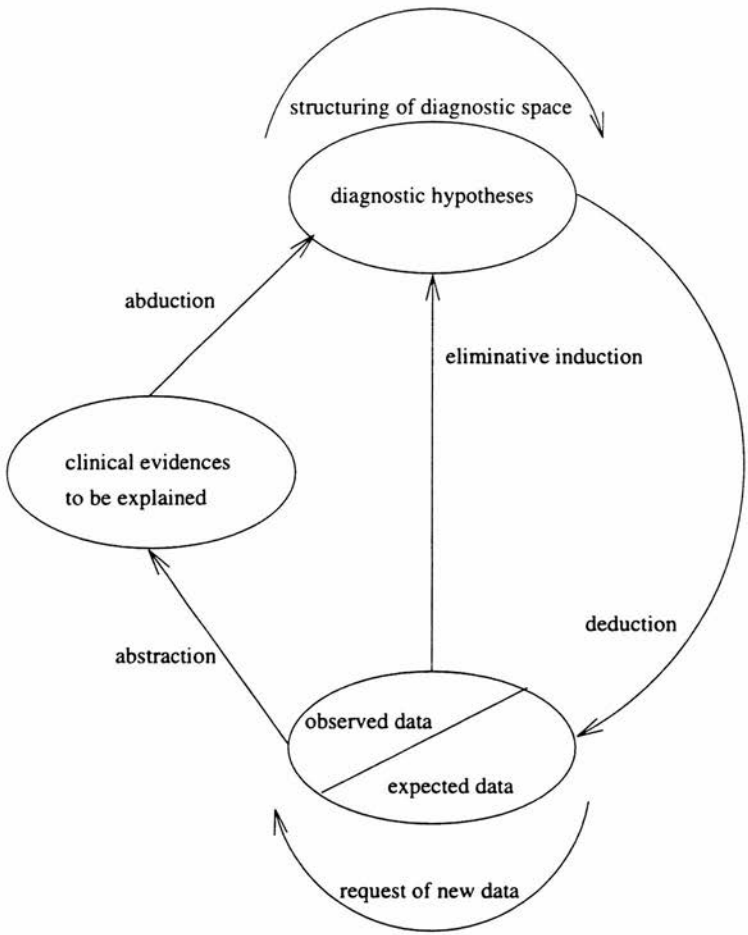
2.3.3 M-KAT

The M-KAT project (part of GAMES-II: a General Architecture for Medical Expert Systems described variously in ([Lanzola & Stefanelli 91], [Lanzola & Stefanelli 92], [Lanzola & Stefanelli 93], [Schreiber *et al.* 93]) aimed to design and implement a knowledge acquisition tool, and, as a requirement on this, to

> "...design a knowledge representation architecture which is declarative and operational, so that it can be easily read both by people and by programs." ([Lanzola & Stefanelli 92] p. 352)

A general diagnostic model was developed (see figure 2.4) which is made up of subparts abstraction, abduction, deduction and eliminative induction.

The M-KAT system built using this model has a 'performance element' which interprets domain knowledge bases and a knowledge acquisition tool for building those knowledge



(Adapted from Lanzola and Stefanelli 1992)

Figure 2.4: M-KAT's model of diagnosis

bases. The performance element does diagnosis in the manner of the diagnostic model described above. The knowledge acquisition tool acquires domain knowledge and uses the diagnostic model to make sure that the domain knowledge is in a form that can be used by that model in a real diagnostic situation.

The performance element has a blackboard architecture. During the diagnostic process it selects which problem solving methods to use (the subtasks) based on the current state of its knowledge. Although MKAT operates with only one high level model of diagnosis, the blackboard architecture of the performance element ensures that this model is flexibly executed.

The system also does task knowledge acquisition as well as domain knowledge acquisition in that the user can customise the performance element by editing any of the control knowledge metarules. User-customised metarules are then always invoked at runtime in preference to the system-defined originals. This customisation of the control knowledge, however, only goes as far as altering how subtasks can be executed. There is no possibility of constructing a new model out of the basic primitives. As with the KADS methodology, the system builder is encouraged to use a given model and twiddle it to their needs.

2.3.4 Problems with modelling applications

NEOMYCIN is still rule based, and so many of the problems outlined in section 2.1.4 will still apply. It also does not make a strict enough distinction between task and domain knowledge in that its task hierarchy has some very domain specific tasks at the bottom. NEOMYCIN's diagnostic model presents a task hierarchy (see figure 2.3), a part-of rather than a subclass hierarchy, so that its model is quite rigid. More seriously, it only really has one diagnostic model and does not allow the building and saving of other diagnostic models. It also has no domain modelling facility.

GUIDON-MANAGE is an improvement for our purposes over NEOMYCIN in that it is specifically aimed at students. Its tasks are at a comprehensible level, not being to abstract or too tied to domain detail and models can be actively built from these subtasks. However, as GUIDON-MANAGE is an interactive environment for 'doing' diagnosis it

does not allow the user to keep and reuse a model. Students do not actually build a model and see how it behaves. rather they choose subtasks to perform one at a time and observe the results. The construction of a complete model would be impossible with this tool firstly because the endusers of the system only have access to a limited band of subtasks and secondly because all the tasks have *arguments*. For example invoking 'pursue-hypothesis' requires the user to specify which hypothesis is to be pursued. This will not work for a system building tool as there has to be a decision made as to the value of the argument (which hypothesis or symptom to pursue, clarify, refine) and this is dependent on the current case and domain data. It also has no domain model building facility. Both NEOMYCIN and GUIDON-MANAGE have a further problem in that they are effectively tied to one domain (meningitis).

DEMEREST is really an a analysis rather than synthesis tool. Its aim is to recognise a student's model of diagnosis rather than to build one from scratch. Again it does not allow us to build and reuse any models. The models of expertise which DEMEREST recognises vary according to the *ordering* combination of the strategies or subtasks and the diagnostic models are not flexible in execution. Like NEOMYCIN and GUIDON-MANAGE, DEMEREST is tied to one area of medicine, in this case orthopaedics.

M-KAT has the advantages that it is aimed at domain experts and that it does both domain and task knowledge acquisition. Models are retained and are reusable. The task modelling facilities are limited however. Although it has subtasks the system builder is not encouraged to do what the authors disparagingly refer to as 'task assembly'. Task knowledge acquisition is restricted to *modification of task execution*. Also, although M-KAT's authors criticise other systems like ROGET, SALT and ONCOCIN for being tied to one task model which is not transferrable to other problems ([Lanzola & Stefanelli 93]). M-KAT itself relies on one diagnostic model, albeit a very general one.

In general, the modelling applications for medicine are better geared than the general modelling systems towards the specific requirements of the medical domain. Their task model primitives are likely to be more recognisable by medical experts and they often incorporate impressive domain modelling facilities. Their major drawbacks are that they

do not allow the saving and reuse of a task model (NEOMYCIN, GUIDON-MANAGE) they are not primarily model *building* systems (NEOMYCIN, DEMEREST) or they do not allow task assembly (MKAT).

2.4 What is missing

Non-modelling methods are inadequate for our purposes because they only deal explicitly with domain knowledge acquisition. The 'diagnostic method' is not really addressed and they tend to rely on some simple inference strategy like backward chaining. There are well-established, if dull, methods for acquiring objects attributes and values, but there are not well established methods for acquiring task knowledge. Most knowledge acquisition methods shy away from describing how they would do task knowledge acquisition, or just do not mention it. Extant modelling methods are not adequate because they are not usable by experts, and indeed the only current tool which implements a modelling method (KADSTOOL) is targeted at and only really usable by knowledge engineers. Modelling applications for medicine are either incapable of building systems, or, like KADS, do not encourage task assembly. None of the medically oriented model based tools we looked at really builds systems.

It is also notable that in most of the models of different types of diagnosis we have looked at, the architecture is essentially static. M-KAT's blackboard architecture is a possible exception in that it does allow flexible task execution. NEOMYCIN is not totally rigid because of the preconditions on metarules but as it is not a modelling tool this does not really help. There are generally component subtasks and an algorithmic description of how these subtasks are to be ordered. Although a new task model can be built in the KADS system, its construction details must also be an algorithmic description. There is no notion of doing a *different* subtask dependent on the status of current knowledge. The only such flexibility is of the type exemplified by a 'stop on success' loop whereby the system continues to do something (like hierarchical classification in the Generic Tasks model) until it has succeeded.

What is required is a real knowledge acquisition tool which allows the building and

subsequent use of models of diagnosis for medical applications. It should be a usable system, not just a methodology, doing both domain and task modelling, be usable by experts, build real systems, not be tied to one task model and build flexible task models. From the systems we have examined here we would like to incorporate

- The model execution flexibility of MKAT
- The domain model building abilities of non-modelling methods and KADSTOOL
- The medicine-specific subtasks of NEOMYCIN and GUIDON-MANAGE
- The model *retention* of KADS
- The model *building* of Generic Tasks
- The simplicity and user-recognisability of DEMEREST
- The domain structuring of template tools

2.5 Summary

Extant methods for knowledge acquisition have their faults. Traditional methods do no task knowledge acquisition. Modelling methods require the services of a knowledge engineer. Modelling applications in medicine either do not build systems or do not allow enough control over the modelling of the diagnostic task. An ideal knowledge acquisition system for medicine would be one which allowed a medical expert to build a system themselves, incorporating both domain and task modelling and result in a real system at the end of the knowledge acquisition process. It would allow both the analysis and synthesis of diagnostic strategies for use by both learners and system builders. In the next chapter we will explore a methodology which makes satisfying these requirements possible.

Chapter 3

A methodology to analyse diagnosis

In this chapter we elaborate the methodology for diagnostic analysis on which we have based the design and implementation of the toolkit. The method we have developed rests on four main elements. First we emphasise the separation of domain and task knowledge in medicine in so far as this is practicable. Second we advocate the use of models to represent both domain and task knowledge. Third we encourage the modular construction of task models from rudimentary diagnostic subtasks. Lastly the representation should be capable of supporting the construction of models which are flexible and responsive, as well as those which are rigid and static. We explore the implications of these four tenets, some indication is given of their particular importance in the field of application and we discuss the design strategy leading to their effective implementation.

3.1 Separation of domain and task knowledge

3.1.1 What it means

We make a fundamental assumption that medical knowledge can be usefully considered to be of two types; knowing what is the case and knowing how to do something. 'Knowing what' will be referred to as *domain knowledge* and 'knowing how' will be referred to as *task knowledge*. We do not claim that this distinction is novel, indeed it is fundamental to most current work in modelling and knowledge acquisition (see Chapter 2). We have only

recognised two types of knowledge as being germane to our analysis and implementation so this should not be seen as equivalent to methods (such as KADS) which divide the 'knowledge space' into more than two levels.

Domain knowledge (also sometimes called declarative, static, or textual) is knowledge about the entities and relationships between entities in an area, such as diseases and symptoms in medicine. This is the sort of medical knowledge that appears in medical textbooks. Examples of domain knowledge might be 'a rash is a symptom of measles', or 'lactose intolerance is common amongst orientals'. Domain knowledge contains no indication of how any problem-solving or other activity might be performed with it.

Task knowledge (also sometimes called procedural, dynamic or problem-solving) is knowledge about how to do something, in this case how to do medical diagnosis. This is the sort of medical knowledge that is often gained by doing it or observing it being done, such as by watching the specialist on ward rounds. Examples of task knowledge might be 'identify which disease type you're dealing with before looking for the specific disease' or 'don't investigate more than one hypothesis at once'. Task knowledge makes no mention of specific domain entities such as individual diseases, symptoms or therapies let alone individual case information. It is concerned only with more generic entities like disease, hypothesis, symptom, which act as variables to be instantiated when particular diseases and symptoms are under consideration.

These two types of knowledge can vary quite independently. Although two practitioners cover the same ground (they are both, say, gynaecologists) their methods of diagnosis may be radically different, due to differences in the way they were taught or personal 'inference style' (see [Balla 85] Chapter 9). Similarly, two practitioners may have been trained at the same school but practise in widely separated parts of the world, covering very different areas of medicine but using the same fundamental diagnostic methods.

This division between domain and task knowledge is not peculiarly apposite to medicine. Many other areas can be subject to the same type of analysis. The knowledge required to be a good teacher can similarly be divided into task and domain knowledge. Someone may be very familiar with say, the principles of Organic Chemistry or English Literature,

but if they do not know how to present that information in a comprehensible manner they will not be a good teacher. Similarly, someone else may be very good at structuring and simplifying information, but have no familiarity with the subject matter. They will be equally ineffectual. The good teacher knows both how to teach (task knowledge) and what to teach (domain knowledge). In medicine too, both types of knowledge are required. Knowing how to do diagnosis is no use unless we also have something to diagnose; diseases to recognise and symptoms to observe. Similarly, knowing about the relationships between diseases and symptoms is not much use unless we also know what to do with that knowledge: unless we also have some strategies to proceed from the identification of symptoms to recognition of an underlying disease. A successful diagnostic support system, like the successful practitioner, requires both types of knowledge. It would know both how to do diagnosis and what to diagnose.

3.1.2 Application areas

In an educational setting the separation of the two types of knowledge allows the identification, and hence the teaching, of generic methods. Generalised methods require less remembering than lots of cases, although they may be more difficult to learn initially precisely because of their abstract nature ([Feltovitch *et al.* 89]). As the amount of information presented increases (and medical students are bombarded with vast amounts of information) such strategies become vital. As one evaluator of our method suggested, task knowledge 'helps prioritise domain knowledge'.

From a knowledge acquisition standpoint the separation allows more efficient use of resources. Experience shows that we can acquire domain knowledge quite easily from experts or textbooks, indeed a widely available variety of sources, with fairly primitive strategies (some of which are outlined in Chapter 2). Task knowledge, on the other hand is difficult to acquire.

> "The knowledge elicitation task is inherently difficult for strategic knowledge, even with traditional methods such as interviewing and rapid prototyping, because strategy is often tacit, and even when it has been made explicit,

> it is not easy to describe it in a form that may be directly translated and implemented into a program." ([Lanzola & Stefanelli 91])

If we acquire the two types separately we can concentrate appropriate knowledge acquisition techniques on the different types of knowledge.

As far as the actual systems built are concerned, the major advantage of the separation is that it enables maintenance, update and customisation. Medical knowledge is surprisingly fluid. Prescribing practice changes as new drugs emerge or old drugs are taken out of circulation. Local epidemiological changes must be catered for; for example the prevalence of malaria in a country may vary widely from year to year, and across geographical regions, dependent on rainfall and other climatic factors. New diseases emerge (like AIDS) and others are eliminated (like smallpox). Diagnostic practice alters according to local variability in access to drugs, laboratory tests, transportation and equipment. The training of health workers can vary widely from one training center to another, emphasising different skills and different diagnostic strategies in procedures like history taking. As mentioned in Chapter 1 (1.1.2), problems can arise with regard to the appropriateness of medical expert systems in the domain or the task knowledge (inference structure). The separation of the two types of knowledge means that we can change the way diagnosis is done without changing the particular instances of diseases and symptoms covered, or change the subject matter the system deals with without changing its diagnostic behaviour. This guarantees that a change in one sector can be effected without jeopardising the integrity of the other. If there is a fault in the reasoning strategy, it is obviously easier to rectify this if the reasoning strategy is explicit. If they are too closely coupled effecting such change becomes unnecessarily expensive, and inefficient if only one part is faulty.

As well as enabling change in a faulty or obsolete system this facility can be used to customise an existing system for different applications. It is possible to reuse the same domain with different tasks, and the same task on different domains. The former type of change would be useful where a system covering the same area of medicine might be implemented with different diagnostic task models to target different categories of endusers; community health workers may benefit from a system with a very different

diagnostic strategy from that appropriate to medical students at different levels of experience. The latter change would be useful for altering the system to be relevant to a different geographic or climatic area, or where the same diagnostic strategy can be used in tools which cover very different areas of medicine. A uniformity of practice can thus be maintained across a suite of systems (e.g. for training purposes) and from an economic standpoint it is not necessary to build each system from scratch if the models can be used modularly.

From the enduser's point of view an explicit reasoning strategy can be useful particularly with regard to ease and naturalness of explanation. The system's questioning can be explained in terms of either task or domain, depending on which is more appropriate. For example the system could ask 'Does the patient have a pink rash?'. A task level explanation of why this question is asked could be that it is trying to differentiate between hypotheses. A domain level explanation might show all the possible diseases with a pink rash as symptom.

3.1.3 Design decisions

We have emphasised a fairly strict division between task and domain knowledge which may be more of a design ideal than a medical reality. It is evident from medical literature and those involved in medical education (on both sides) that the division is not as strict as would be computationally ideal (this is explored more fully in Chapter 6). However, it is a distinction which is recognised, valued and seen to be of use particularly for teaching purposes ([Rodolitz & Clancey 89]).

Ideally, domain knowledge should be in a format which is usable in multiple activities, so that e.g. in medicine it might be used by a diagnostic problem-solving method, but equally well by an information browser or an epidemiological data gathering facility. The 'Oxford System of Medicine' project had precisely this as one of its goals, being a generic medical information system

> "... with the versatility to support information retrieval, data management and decision support facilities for medical practitioners" ([Glowinski *et al.* 89])

Ideally, task knowledge should be capable of operating with many different domains so that e.g. a diagnostic method would be equally effective working in the domain of haematology or psychiatry. To these ends we decided to have two separate tools, one of which concentrates exclusively on task knowledge and one of which is devoted to domain knowledge.

3.2 Modelling domain and task knowledge

3.2.1 What it means

(Contemporary modelling techniques are further outlined in Chapter 2) In building a diagnostic support system one of the best ways to make the tool embody the desired knowledge is to model that knowledge. There are many types of models, many things that can be modelled, and many purposes for building a model. 'Model' covers everything from structural representations such as maps and diagrams through mathematical and statistical models to symbolic conceptual models of processes. John Kunz in [Kunz 88] provides a useful breakdown of model types.

- *Physical models* are physical representations such as scale models. They are used for anticipating the structure and predicting the behaviour of systems.
- *Informal symbolic models* are system descriptions based on public knowledge. They are used to analyse and predict the behaviour of systems.
- *Diagrams* emphasise the structure of a system, the function is usually implicit. They are primarily explanatory devices.
- *Formal mathematical models* describe the functional behaviour of systems, the structure is only represented implicitly. These abstract predictive models are used to organise problem analysis.
- *Heuristic models* symbolically describe the behaviour of systems based on experts' descriptions. There is no explicit representation of the structure of modelled system.

- *Formal symbolic models*: 'explicitly represent both the structure and functional behaviour of the modeled system'. They describe both the problem domain and the behaviour of the system. Importantly, they 'emphasise the relation between input, output, and internal states of the model which correspond to the modeled system'.

Most model-based reasoning is based on formal symbolic models, and this is the strategy we adopt here. The main characteristics of such a model are that it is

1. *idealised:* The real world is noisy, complex, dangerous and vulnerable whilst a model provides an idealised, simplified representation without the noise.
2. *explicit:* in that there must be some direct, non-arbitrary and recognisable correspondence between the model and the thing modelled.
3. The model covers both *structure*, the concepts, concrete objects, attributes and relations, and *function*, what is defined in the theory of the process.
4. Models are *generic* 'Model based reasoning implies use of generic reasoning methodologies which apply for any class of structures and behaviours.' ([Kunz 88])
5. *Control* is separated from the structure and function description '...the model-based approach is characterised by use of a domain model which explicitly represents structure and function of the modeled system separately from the control of the reasoning process' ([Kunz 88]).

In keeping with our division of medical knowledge into domain and task knowledge, we model both structure and function separately (corresponding to structure and function of a 'diagnoser's' knowledge respectively) resulting in a domain model and a task model.

A domain model is one representation of an area of static knowledge, in this case knowledge about diseases and symptoms and the relationships between them. There can be many domain models which cover the same material but present it in a variety of ways, just as there can be many textbooks which cover the same material but present it in a variety of ways. Two domain models built by two experts in the same area might be

very different e.g. in a particular domain model the disease object 'measles' is a type of 'infectious diseases', which are all 'diseases'. One of measles' symptoms is 'red rash starting behind ears'. This is a type of 'red rashes' which is a type of 'rashes' which are all 'symptoms'. In another domain model, built by another expert, 'measles' might be classified as a 'childhood disease' and its symptom be 'red spots', a type of 'spots'. These differences have no implications for the way the experts perform their diagnoses.

We are particularly interested in task modelling of medical diagnosis, although the system we describe also provides facilities for domain modelling, and in how to effectively build task models which will function adequately in decision support. Although a complete specification of the domain contains all the information necessary to make a diagnosis, all the associations between symptoms and diseases, the decision space in any domain will generally be too great for blind match, search or any other simple inference mechanism to be useful or efficient. An inference method or reasoning strategy is needed to pull out the domain information necessary for the particular case under consideration. This is what the task model supplies, in an explicit format. A task model is a specification and representation of that how-to knowledge, a method of using the knowledge in the domain to solve actual problem cases. Again, two task models built by two different experts might be very different dependent on the way they have learned to do diagnosis, the way they wish to pass it on or the way they wish a diagnostic system to behave. For example, one expert may like to gather as much data as possible on a case before exploring possible explanations. Another might wish to gather more data only in the pursuit of particular hypotheses. These differences in strategy have no implications for the way in which the two experts organise diseases and symptoms in any domain.

3.2.2 Application areas

Modelling is a useful method both from the point of view of the system builder, be they knowledge engineer or domain expert, and of the enduser of the system being built.

Because it is *idealised* (like what is modelled, but simpler) the model is easier to work with than the system modelled. An idealised representation contains only what is important in the system so is particularly appropriate for teaching purposes. Users can test things

out without hurting themselves, or causing problems in the real world, so models are good for teaching in medicine where students are not let loose on real patients until they have some experience. The creation of a model is often regarded as a good way of exploring the characteristics of the process being modelled, both to understand the process and, possibly to improve on it.

> "The modeling approach seeks to model the way we perform tasks that we recognise as requiring intelligence, and it seeks to elucidate the mechanisms we employ in our own solutions to these problems...The modeling approach to AI therefore views the implementation of computerised models as a key technique for understanding intelligence. In addition, these models often suggest novel mechanisms that may become part of the conceptual theory itself." ([Rothenberg 89])

Building the idealised version encourages an identification of what are the important bits in e.g. diagnosis, and this can expose flaws which might not be apparent in the real 'dirty' version. This characteristic is particularly useful in analysing problems that are poorly understood or ill-specified (e.g. disaster modelling where significant expertise does not exist).

Because it is *explicit* (there is an obvious mapping between the structure of the original and the structure of the model) system builders do not have to learn arbitrary coding conventions, ensuring ease of programming and design. Knowledge contained in a model can be changed or updated relatively easily because it is explicitly represented. The correspondence with reality ensures that it is possible to readily identify that part of the knowledge structure which is to be replaced or modified. Since the knowledge is explicit to both developers and users, both can suggest changes. This encourages the possibility of customisation by experts themselves, so that they can construct a model corresponding to the way they perform the task (or think it ought to be performed) themselves or ensure diagnostic behaviour that is familiar to the enduser. Explicit representations make for more obvious explanations. Explanation in terms of task aims and goals is generally easier to interpret than that given in terms of rule-tracing as there is a closer correspondence

to recognisable human activities When the system fails, a model which corresponds explicitly to what is modelled, a human diagnostician, ensures graceful degradation as it is more likely to fail in an understandable manner.

Because it is *generic* the model can potentially be used for multiple applications (design, diagnosis, simulation, scheduling, throughput analysis, browsing) and also for multiple users. Information can be presented at different levels of detail and viewed from different perspectives.

Because there is *separate control* information the model can avoid many of the pitfalls of rule-based systems, particularly those resulting from the mixing up of control and other rules such as in explanations.

3.2.3 Design decisions

A domain model in our representation is a hierarchical structure of disease and symptom data objects. There are also relations between these objects, such as that a certain disease causes a certain symptom, or that a certain symptom is very rare. The general structures in a domain model reflect the way that the system builder *classifies* diseases and symptoms.

A task model is a representation of one way of doing a task (in this case medical diagnosis). In our representation a task model consists of unitary *subtasks*, which will be explained in the next section, and information on how and when these subtasks are to be executed.

3.3 Building models from subtasks

3.3.1 What it means

Task models of diagnosis are often described in terms of smaller subparts, or subtasks, which can be modularly combined to form a complete diagnostic task. For example, one simple-minded way of doing diagnosis might be to

- collect the presenting symptom (what the patient presents with) and find all the diseases that might have that symptom.
- choose the first of those diseases and ask about one of its other symptoms
- find all the diseases that have both the first symptom and the user's answer to the second symptom question
- continue thus until only one disease conforms to all the evidence

The above ordered list of diagnostic subtasks could serve as a (simple-minded, as we said) model of diagnosis. Using such subtasks there are a multitude of other models of diagnosis which could be similarly defined.

A subtask is a primitive inference, performing some function over domain knowledge, which is

1. *atomic*: in that it is not decomposable into smaller bits,
2. *modular* in that it can be combined with other subtasks
3. *unitary*: in that could be completed and make sense on its own,
4. *generic*: in that it is reusable in many task models and may even may be reusable in other problem solving tasks
5. *recognisable*: as a part that could help to realise the general goal of the task although it need not be peculiar to that task.

It is no new idea to describe problem solving strategies in terms of smaller subparts, or subtasks, which can be modularly combined to form a complete task. The two main current knowledge acquisition paradigms, from the KADS group in Europe and the Generic Tasks group in the USA, are of this type. The 'expertise' model, a model of the expert's problem solving behaviour, is central to the KADS methodology. Although KADS emphasises model *refinement* rather than model *building*, where the system builder is encouraged to choose one of a selection of interpretation models from a library, the

methodology also allows the construction of a task model from primitives if none of those in the KADS interpretation model library are appropriate. (see [Hickman *et al.* 89] for an accessible description of the KADS methodology.) The equivalent of a subtask in the KADS methodology is a 'knowledge source' which

> "...performs an action that operates on some input data and has the capability of producing a new piece of information ('knowledge') as its output. During this process it uses domain knowledge. The name of the knowledge source is supposed to be indicative of the type of action that it carries out." ([Wielinga *et al.* 92a], p. 19)

Generic tasks are also building blocks for constructing high-level compound tasks. They are of a coarser granularity than the 'knowledge source' primitives of the KADS approach. They are characterised by their input and output, the control regime they implement and the knowledge constructs they use. (See [Chandrasekaran 88] for a detailed outline of the Generic Tasks methodology).

3.3.2 Application areas

Building a task model out of subparts is useful for the same reasons that any sort of modularity is useful. In general, a modular system is like a high-level programming language, which is easier to use as the system builder need not get bogged down by low level implementation details. If the implementation language is at too low a level, it can force the system builder to express their knowledge in an alien format. For example it is difficult to get experts to express their knowledge in production rules and the search for such items often obscures the more structured aspects of their knowledge. A building environment without code is particularly desirable if we envisage experts using the system themselves, as we do not want to turn experts into knowledge engineers or programmers. Potentially, this part of system design and construction can be done by the experts themselves, if they recognise the subparts, obviating the need for the expensive knowledge engineer. In the toolkit we describe in Chapter 4 all subtasks are designed to be recognisable components of the expert behaviour being modelled. Subtasks are

not just useful programming primitives but can actually reflect the methods used by human problem-solvers. If the experts can recognise the subtasks it is likely that these are primitives which they use, or could use, in practice and our evaluation results (see Chapter 5) suggest that the chosen subtasks were largely recognisable by the experts.

Because the units out of which the task model is built are self-contained, explanation need not be low-level. If the elements are also identifiable by the enduser they can be treated as atomic black boxes which do not require further explanation or decomposition. It is likely that endusers may be able to recognise small parts of a task where they would not recognise a complex whole.

Since subparts are potentially reusable a task model need not be built from scratch each time. The model can be easily changed by removing, replacing or adding subparts.

3.3.3 Design decisions

The software we describe provides facilities for building a task model out of small modular subtasks which are independently identifiable elements of diagnosis. It is specifically geared to medical diagnosis, so the subtasks available in the current system are elements of the diagnostic process. In principle, however, the subparts are generic and can be used as building blocks in the assembly of other types of task model, such as for planning or design. We have identified a number of these subtasks which can be modularly combined to form a complete diagnostic task. This combination of subtask modules, with constraints on their method, timing and ordering of operation constitutes the basis of a diagnostic task model.

We do not provide facilities to compose new subtasks, although this is a feasible extension. We felt that users would be unlikely to want this facility at a stage where the entire method is novel, and it was more important to concentrate on building models with a limited set of familiar elements. This was borne out by the evaluators, one of whose comments sums this up:

> "It's difficult to think up subtasks because they're things you think are common-sense. If you present people with them and they recognise them

that's better and they'll just say 'oh that's obvious'."

The sources which provided the inspiration for our limited set were

- modelling literature (KADS and generic tasks)
- other work on modelling in medicine (Neomycin etc.)
- psychological investigations of medicine ([Evans & Patel 89])
- invention

although the any faults in subtasks included in the TOMKAT system should be seen as being entirely peculiar to the implementation of that system.

3.4 Building responsive models

3.4.1 What it means

Any construction is, of course, not just a jumble of parts. It must have some structure. The task model too must consist not just of the composing subtasks, but a description of how these subtasks fit together during the process of a real diagnostic problem-solving.

Such a description can be primarily static, describing the explicit sequencing of subtasks relative to each other. We call such an ordered list of diagnostic subtasks a *rigid* task model. Another type of description is more dynamic and flexible, resulting in a model which executes its subtasks in response to incoming data. We call such a model a *responsive* task model. These two methods are somewhat antithetical in their effect, as the more rigid structuring is put into the task model the less responsive can its problem-solving behaviour be, or the less we can control its behaviour.

3.4.2 Application areas

Are there identifiable characteristics of expert diagnostic strategy compared to novices, aside from mere accumulation of extra factual knowledge? Alpay ([Alpay 90]) felt that

previous research had ignored, or not uncovered the differences between novice and more experienced practitioner's diagnostic behaviour. He analysed this difference in detail, characterising the development from novice to expert in terms of different combination of elements ('strategies') which act very like our subtasks.

> "...one assumption which prevails in the medical problem solving literature is that novice as well as more experienced physicians use the same reasoning processes...While medical students as well as experienced physicians may use similar reasoning strategies, they do not always combine them in the same way." ([Alpay 90] p. 275)

Earlier studies of expert and novice diagnosticians suggest that experienced and more effective diagnosticians employ an adaptive method ([Leaper *et al.* 73]) which is responsive to incoming data, whereas inexperienced diagnosticians follow a very static algorithm and use an 'ordering rather than structuring' method ([Ramsden *et al.* 89]). In building a decision support system it is obviously better to try to reflect this adaptive behaviour of successful and experienced practitioners, something which the responsive task model makes possible.

Although the rigid model will eventually succeed, it is blind to its own current state of knowledge. The responsive model is able to modify its problem solving behaviour on the basis of its own view of its current state of knowledge. This awareness of its own status enables it to respond appropriately and efficiently to incoming data rather than grinding through a prearranged sequence of actions. The subtasks used, and thus the diagnostic algorithm, can be changed dynamically as conditions demand. This also minimises 'thrashing' (where the system uses pointless and/or repetitive strategies).

3.4.3 Design decisions

We have developed a method of constructing responsive task models which uses the notion of *constraints* to specify system control. A responsive model is built by specifying characteristics of each subtask which will guide when and how they can operate. For example, consider a subtask called 'group' which groups diseases into types. It might be

sensible to say 'group your possible diseases into types when they are all very specific diseases'. What we are doing here is putting a *constraint* on the 'group' subtask as to when it should operate, in effect providing behavioural guidance for a controller which dynamically orders subtasks dependent on incoming data and the guidance information set by the system builder.

In fact, any statement about the ordering of the subtasks in a model can also be thought of as a constraint on when that subtask can operate. If there is a task called 'collect-symptom' which should always be done first, this is tantamount to placing a *constraint* on when that subtask should operate. This is what we call an 'algorithmic' constraint, because an ordered list of subtasks gives an *algorithm* for diagnosis. The constraint we put on grouping above is a 'whenever' constraint, because we do the subtask *whenever* a certain state obtains.

A third type of constraint which has been incorporated in this design is the *how* constraint. This specifies how a subtasks operates when it fires. A *how* constraint is generally an instruction as to how a choice is to be made, such as the instruction to perform the subtask 'choose-hypothesis' by choosing the most common hypothesis. This will be further outlined in Chapter 4.

The complete task model thus consists of the diagnostic subtasks and a set of constraints. The constraints act as instructions to a central controller as to when, how and in what order it should execute these subtasks.

3.5 Summary

Diagnostic methods are difficult to get hold of, although if we can get hold of them they have great potential both for teaching and in building better diagnostic support systems.

Medical knowledge can usefully divided into domain and task knowledge. This division is helpful for teaching generic strategies, allows efficient application of knowledge acquisition techniques and, in the systems being built, enables maintenance, customisation and explanation.

Modelling is currently a popular knowledge acquisition technique which allows ease of system building, modification, design, verification and explanations. Task models in particular produce more efficient systems and allow the system builder control over system behaviour. Our method emphasises the construction of both domain and task models.

Task model building with subtasks makes for a naturalness of representation which permits the experts themselves to be system builders. Subtasks also result in simpler explanation facilities and, being generic, can be used in different models. Our task models are built from a limited range of atomic subtasks.

Task models built from subtasks can be either rigid (of fixed order) or responsive (of variable order). The responsive models more accurately reflect the behaviour of experienced practitioners and are more effective in dealing with real world data. We allow for specification of rigid and responsive elements with the use of three types of constraints on subtasks.

Chapter 4

TOMKAT

This chapter describes the implementation of the methodology described in Chapter 3 in a toolkit called TOMKAT (Task Oriented Medical Knowledge Acquisition Toolkit). TOMKAT has been built in Common Lisp using interfacing and object-handling facilities provided by Intellicorp's KEE (Knowledge Engineering Environment).

TOMKAT has two main areas of application. It is first a prototypical knowledge acquisition toolkit for building systems which do medical diagnosis. The intended users of the toolkit would be experts in the medical area, although not necessarily experts in a specialised medical area. The intended users of the systems which these experts build with the toolkit would probably be health workers with less medical training. The toolkit provides the experts with facilities to build systems themselves which they can tailor to the needs and abilities of their chosen endusers. Its second potential application area is in medical education. As outlined in the Chapter 1, the uncovering and exploration of the diagnostic method is of use in characterising good diagnostic practice for the benefit of those who have to learn how to do it. As the NEOMYCIN and GUIDON-MANAGE experiments showed ([Clancey & Letsinger 84] and [Rodolitz & Clancey 89]), an environment for exploring the practical effects of a diagnostic theory is also of direct value to trainee practitioners. The toolkit provides facilities for both constructing and exploring the behaviour of different diagnostic methods.

The toolkit consists of two tools; a *task model tool* and a *domain model tool*. The task model tool is what the system builder uses to produce a *task model*. The domain model

tool is what the system builder uses to produce a *domain model*. When a diagnosis is 'run', a *control* mechanism chooses and executes subtasks on the basis of the instructions of the system builder incorporated in the task model. These subtasks consult the domain model and the enduser and update the content of various diagnostic objects, which constitute the current *case model*. Figure 4.1 shows diagrammatically the relationships between these constituent parts and the system builder and end user. The sections following describe how the methodology of Chapter 3 has been made concrete and the implementation of these parts in detail.

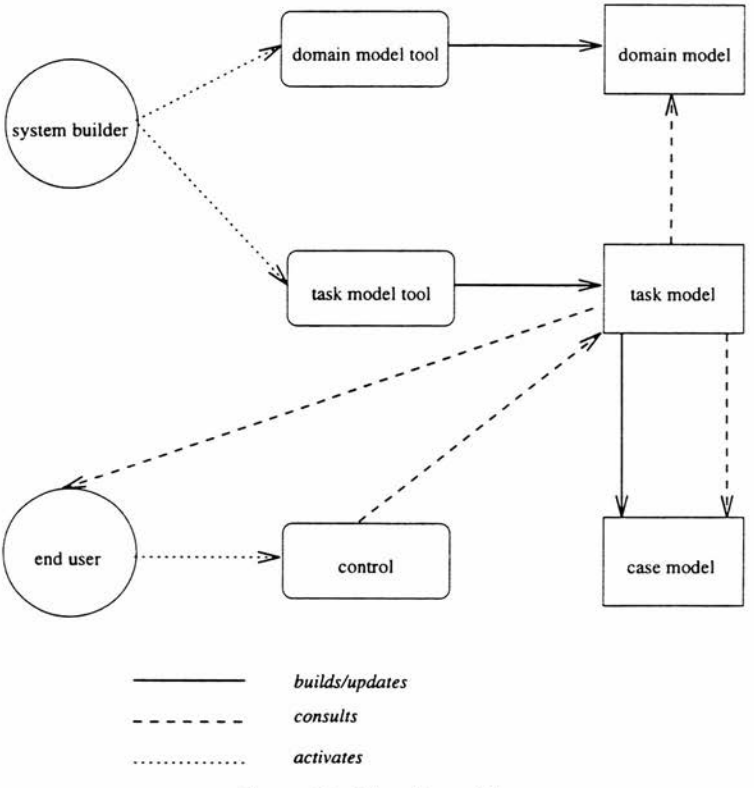


Figure 4.1: Using the toolkit

4.1 Implementing the methodology

4.1.1 Separation of domain and task knowledge

In Chapter 3 we emphasised the separation of domain and task knowledge for teaching generic strategies, allowing the efficient application of knowledge acquisition techniques and enabling maintenance, customisation and explanation of the systems being built. This implementation requires a strict division between task and domain knowledge, our aim with the implementation being to provide an environment for representing a diagnostic strategy or strategies which could be used with a very large set of domain models. We also wished the environment to be able to produce domain representations that could be used with any diagnostic strategy. We have thus made a quite substantial assumption that diagnostic strategy is *not* influenced by the domain of application and that domain and task knowledge are independently specifiable. We also felt that from the point of view of satisfying pedagogic aims, it was important to emphasise a fairly strict separation. Part of the reason why medical students find diagnosis obscure is that it is not taught as a subject per se, and examples are always tangled up in domain detail. Providing a clear strategy gives the learner a chance to concentrate on what aspects of the diagnosis are crucial, so that they can replicate it in situations where particular case detail may be different.

Thus we have two entirely separate acquisition tools; one for capturing task knowledge and one for capturing domain knowledge. The two methods used are different and appropriate to the two types of knowledge. The domain model tool is basically a dedicated object hierarchy building tool and the task model tool is a mechanism for eliciting constraints on control behaviour and a critiquing facility.

4.1.2 Modelling domain and task knowledge

In the previous chapter we showed how modelling allows for ease of system building, modification, design, verification and explanations, and how task models in particular allow the system builder control over system behaviour. Our two acquisition tools are both modelling tools. The system builder constructs a model of the domain knowledge

structure using the domain modelling tool and a model of how diagnosis is to proceed using the task modelling tool.

A domain model is a convenient way of organising the sort of static factual medical knowledge which appears in medical textbooks. It is a description of concepts, objects and relations in the domain of application, the relations in particular supplying much of the knowledge structure. The domain model tool is a tool for entering and maintaining that declarative knowledge of the expert, where the task model tool deals with the procedural knowledge. All domain models constructed with the domain modelling tool are object classification hierarchies of diseases and symptoms. Disease and symptom classifications are not universal in medicine, hence there are facilities to create new classification hierarchies and arrange them in new ways. There are, however, potentially useful extant hierarchies and for this reason we have incorporated a general medicine domain model library (loosely based on that of the Oxford System of Medicine, [Fox *et al.* 87]), from which subhierarchies can easily be borrowed and changed to adapt to the application area being addressed. An example of a partial disease classification hierarchy is illustrated in figure 4.2. In addition to the classification hierarchies of disease and symptom objects there are also relations between these objects more complex than the parent/child classification relationship which structures the hierarchies. The most important of these, for the purpose of doing diagnosis, are the complementary 'has-symptom' and 'symptom-of' relationships between diseases and symptoms which are part of any domain model built with this tool. For example, the disease object in the hierarchy of figure 4.2 'measles' has 'rash' as one value of the slot 'has-symptoms'. Similarly the symptom object 'rash' in the symptom hierarchy of that same domain model has 'measles' as one value of the slot 'symptom-of'. System builders can also invent new relationships or borrow them from the domain model library, such as that a certain symptom rules out a certain disease, or that a certain symptom is very rare. The effective utilisation of these new relationships would await the creation of new subtasks (see below) which explicitly mentioned them. At present, available facilities in the task model tool only make use of the classification relationships and the 'has-symptom'/'symptom-of' relationships. The implementation details of the domain model tool are further outlined in sections 4.3.1 and 4.3.2.

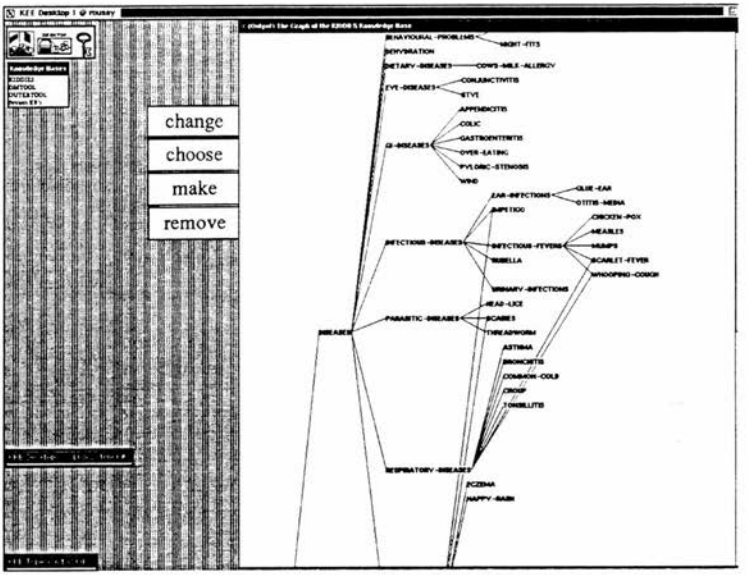


Figure 4.2: Example of disease hierarchy

A task model is a representation of one way of doing diagnosis, and in our toolkit we have implemented a method of building such a representation out of unitary subtasks, things like getting hold of the presenting symptom, focusing attention on a sub-group of diseases, or ruling out a specific disease. A task model built with the task modelling tool consists of subtasks and instructions about how, in what order, and under what circumstances these subtasks are to be executed. The modelling tool for building task models provides facilities for manipulating these subtasks and expressing how and when they are to be executed.

4.1.3 Building models from subtasks

In the previous chapter we described how task model building with subtasks provides a representation which permits the experts themselves to be system builders. Subtasks also result in simpler explanation facilities and, being generic, can be used in many different models. Our task models are built from a limited range of reusable atomic subtasks.

A subtask is basically a procedure; a piece of code which does a bit of diagnosis. It must be something that can be done on its own, be a recognisable sub-part of the diagnostic process and have an identifiable, useful effect. Some tasks collect new data and some do not. New data is obtained either from the enduser, by asking, or from the content and structure of the domain knowledge base. Subtasks always cause some change in the current *case model* (see section 4.4.2) in that they alter the status of the objects which hold the current case data.

As alluded to in the description of domain modelling above, many of the inferences embodied in the subtasks make use of the relationships between objects in the domain model; both the hierarchical structuring (parent/child) relationships and relationships such as the 'has-symptom' and 'symptom-of' relationships. For example the subtask 'group' relies on the hierarchical structure of the domain to find higher categories into which hypotheses fall, whilst the subtask 'differentiate' looks at the 'has-symptom' relationship between hypotheses and symptoms to choose which symptom to ask about next.

Because the notion of task knowledge may be unfamiliar to the expert, we do not require them to identify such knowledge and enter it cold. Rather we present limited options to choose from. The expert thus does not need to invent subtasks, only to recognise them, hence the stress on subtasks being *recognisable* parts of the diagnostic process. The subtasks available in the task modelling tool are all elements of the medical diagnostic process but in principle they are generic and could be used as building blocks in the assembly of other types of task model, such as for planning or design. The sources of our subtasks, details of their implementation and descriptions of their effects, are detailed in section 4.4.1 below.

4.1.4 Building responsive models

In the previous chapter we described how task models built from subtasks can be either rigid (of fixed subtask ordering) or responsive (of variable subtask ordering), the advantages of building responsive models being that they reflect the flexible behaviour of experienced practitioners and are more effective in dealing with real world data. A task model consists both of subtasks and instructions about their execution. When a diagnosis is 'run' the execution of subtasks is effected by a *control* mechanism (described in section 4.4.3). Part of the task model is a collection of instructions to that control mechanism on how to choose which subtask to do next and how to execute it once chosen. The task modelling tool provides facilities for producing both rigid and responsive models, and those which mix elements of both, by issuing such instructions to the control in the form of three types of *constraints* on subtasks.

The system builder can firstly specify how the subtasks are to be executed. For example, a selection subtask like 'choose-hypothesis' (which selects one of a group of hypotheses as described in section 4.4.1) might be executed by choosing to look at the most common disease first, or the most dangerous. A subtask which involves a choice of symptom, like 'confirm-hypothesis' (which chooses a symptom of the hypothesis and asks the user about it as described in section 4.4.1) might be governed by whether the user would prefer to look first for symptoms that are pathognomonic, or that do not require laboratory tests. The implementation of how-constraints is described in section 4.4.4.

The system builder has a second mechanism for control over the diagnostic procedure in being able to place constraints on the overall algorithmic structure of diagnosis. This may take the form of constraints on individual subtasks about where or how often they can occur during a diagnosis, such as stipulating that the control should 'do collect-a-symptom once only', or co-occurrence restrictions on sets of subtasks such as requiring that the control 'do differentiate after group'. In principle, the system builder could specify a completely rigid order in which the subtasks must be executed, in effect completely overriding the choice behaviour of the control. This would be achieved with the use of the algorithmic constraints 'first-task' and 'strict-order', the latter of which specifies that one named subtask must immediately follow another. Such a specification would constitute a rigid task model.

The last type of constraint mechanism relies on the behaviour of the control mechanism which runs the diagnosis. The control is essentially a simple algorithm which chooses which subtask to execute next. It does this by prioritising subtasks, firstly on whether they are marked as being something which *must* be done next because of the imposition of an algorithmic constraint (for example if we are at the beginning of a diagnosis and we have set the algorithmic constraint that 'collect-a-symptom' must be the first task, the control must choose that subtask) and secondly on the basis of our last constraint type, the 'whenever' constraints, being satisfied. These whenever constraints make requirements on the current state of the diagnostic system's knowledge, the case model, which must be met before a subtask can fire. This current knowledge about the case under consideration is held in the case model in the *diagnostic objects*. Diagnostic objects are described in detail in section 4.4.2 below. The control looks at the status of relevant diagnostic objects, like the focus and differential, matches the overall current state of the system with the state requirements of available subtasks, and initiates a subtask. This mechanism is implemented by attaching simple adverbial clauses to the subtask. For example a focusing subtask like 'group' might be appropriate when the number of possible hypotheses is very large, so we would attach the phrase 'when differential getting-big' to the subtask 'group'. We might wish to execute an exploratory subtask like 'elaborate' only when the focus is very general, and so on. Again there are limited options on

which diagnostic objects can be involved, and what states they can be in, for the system builder to choose from. We do not yet permit the definition of new state descriptions for setting 'whenever' constraints, although this is a feasible extension. The setting of these 'whenever' constraints is what allows the specification of a responsive task model. The responsive task model ensures that the behaviour of the system is affected by current knowledge contained in the case model rather than only by a predetermined ordering of subtasks.

Lastly the task model also holds a further 'instruction' for the control mechanism; when it should stop. This is known as the success criterion and is implemented exactly as the 'whenever' constraints, being defined as a state requirement on a diagnostic object in the case model. This will usually be the differential ('when differential one-left' is a common success criterion) but potentially it could be any diagnostic object.

Details of the specific 'how', 'algorithmic' and 'whenever' constraints implemented in the task model tool are in section 4.4.4 below.

4.2 The outer tool

The outer tool consists simply of facilities to load and delete the task model tool, the domain model and the users' tool.

4.3 The domain model tool

The facilities in the domain model tool are those which enable domain structuring through the creation of the object hierarchy, those for creating and filling in relations between objects and the domain model library which is a resource of ready-made objects and relations. The ability to change and remove objects and relations also allows domain model maintenance. The functionality of the domain model tool is represented diagrammatically in figure 4.3.

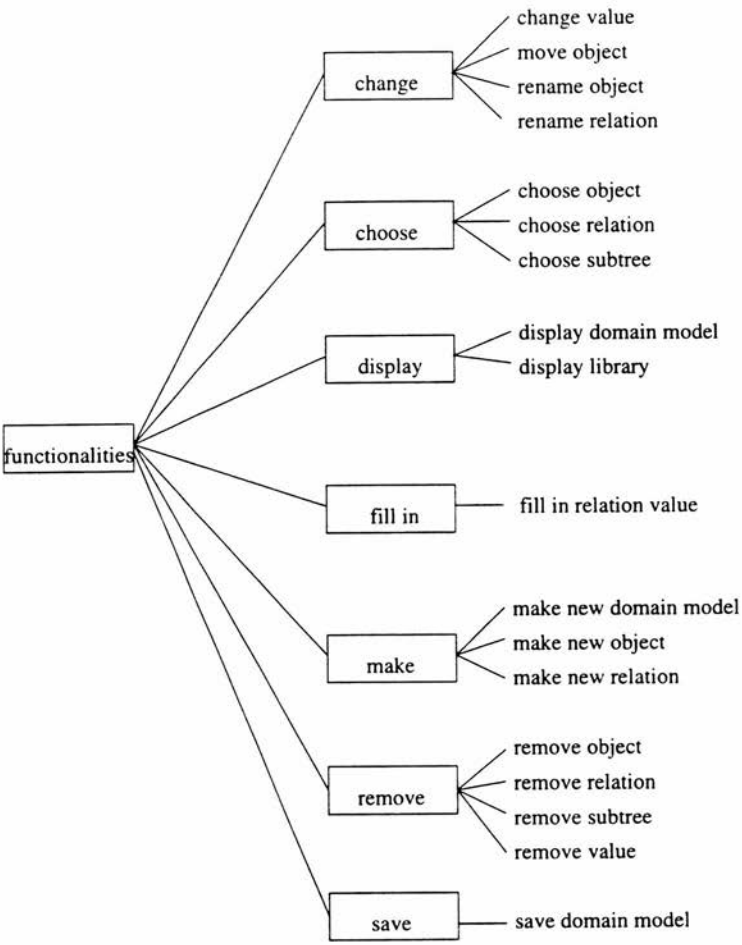


Figure 4.3: Structure of the domain model tool

4.3.1 Domain structuring

The domain structuring facilities allow the expert to build a skeletal domain model from more primitive elements, analogous to the subtasks in the task model tool. As subtasks are part of the larger diagnostic procedure, so elements like 'diseases', and 'symptoms', are parts of larger medical domain models. A domain model consists of a hierarchy of diseases and symptoms, with associated relationships between these diseases and symptoms. All domain models built with this tool are hierarchies of diseases and symptoms, but the names 'diseases' and 'symptoms' are to some extent arbitrary, although they are appropriate to medical applications. The hierarchies could equally well be called 'faults' and 'findings', and indeed might be in, say, a troubleshooting system. The only requirement is that the same type-names be used in the domain model as are used by the task model. Our subtasks are expecting to be able to look up, for example, all the children of, or the values of slots on entities under the categories of 'diseases' and 'symptoms' so our domain models are also organised under these headings.

When any domain model is set up with this tool it is 'seeded' with the top level categories of diseases and symptoms. All new objects created in the domain model must be descendants of these two types. Whenever the system builder creates a new object, or imports one from the domain model library, they are asked which top level category it falls under, and as the subhierarchies of diseases and symptoms become larger and more complex the system builder is assisted in placing new objects within them by a series of cascading menus.

4.3.2 Relations between objects

As well as choosing which elements have to be specified in the domain model, the system builder must outline the characteristics of these objects which will be used in diagnosis.

Every domain model built with this tool is initially seeded with its top level object types, and these types have certain built-in characteristics. 'Diseases' has a relation (implemented as a slot on the diseases object) called 'has-symptoms' which can take any number of values since a disease may have many symptoms. 'Symptoms' has a

relation 'symptom-of' which can also have any number of values as a symptom can be symptomatic of many diseases. As all new objects created in the domain model must be descendants of these two types, through inheritance, any new disease will have a 'has-symptoms' slot and any new symptom will have a 'symptom-of' slot. These two relations are reciprocal, in that an addition to a 'symptom-of' slot (such as the value 'measles' in the object 'rash') will cause the appropriate value to appear in the corresponding 'has-symptoms' slot on that disease (so that the value 'rash' will appear in the 'has-symptoms' slot on the object 'measles'). This feature is implemented through KEE's *active values* and not only cuts down domain model building time but also ensures consistency in the domain knowledge base.

It is possible to create other new relations between objects, or one-place relations on individual objects, other than the the symptom-of and has-symptoms relations. At present, all the subtasks implemented in the task model tool use only these relations and the structural (child, parent, descendant, ancestor) relations in the domain model. However, the mechanisms exist to exploit other relations. Many of these could be done by the construction of further 'how' constraints (see section 4.4.4 below) on subtasks in the task model. For example, setting a constraint on a subtask that needs to compare the relative values of 'dangerousness' of diseases dictates that there must be such a characteristic for every disease in the domain model. Possible features might include temporal information (on the course of disease), probablistic information (weighting the causal links between diseases and symptoms) or epidemiological information (about the frequency of a disease in specific populations). It should be noted that we have not made explicit provision for the specification of disease *process* descriptions, an important aspect of medical domain knowledge. It seems feasible that disease process information could be incorporated within the basic disease and symptom hierarchies through more complex relationships but we have not tested this and our task modelling facilities do not have no requirements for disease process information. A system builder might include relationships on diseases such as 'acute-symptom' and 'chronic-symptom' with corresponding relationships on symptom objects. Details of how to create objects and relations, import parts of the domain model library and change aspects of the domain model are contained in

Appendix C which describes the use of the both the tools.

4.3.3 Domain model library

The domain model library is, like all other domain models, a classification hierarchy of disease and symptom objects with the usual 'has-symptoms' and 'symptom-of' relations. All the diseases and symptoms in this knowledge base can be 'borrowed' when building a domain model. In fact whole subtrees of the hierarchy can be directly copied. A sufficiently broad-based library can thus shorten the time needed to build the domain model considerably for any particular area as, even if the structures in the library do not directly correspond with what is required they can provide a starting point for modification.

The domain model library also holds a hierarchy of relations; 1-place relations such as 'dangerousness' or 'frequency' and 2-place relations like 'rules-out' or 'suggests'. These relations can also be imported into the domain model being built, although at present no available subtasks or constraints use them. The domain model library's general structure is loosely based on the disease and symptom categories of the Oxford System of Medicine, OSM ([Fox *et al.* 87] and [Glowinski *et al.* 89]). This is a large database consisting of about 6000 objects and 30 relationships which are arranged into about 9000 core facts with 6-7000 further derived facts. The database covers the wide domain scope of general practice and the information is either taxonomic ('diseases generalises infections') or causal ('abdominal pain can be caused by appendicitis'). The OSM database is designed as a core of medical information which can be used by multiple agents such as a browser or a diagnostic module without being specifically tied to any one of them, much as our domain models are design to be used by multiple task models without being committed to any particular one. The domain model library we use is, of course, not nearly as large as the OSM.

Although only one, general, library model currently exists to borrow from, it is feasible that multiple library models could be incorporated, which would considerably reduce the time taken to build a domain model. Currently whole domain knowledge bases can be loaded into the domain modelling tool, but only the general library model can be used

as a resource pool from which to borrow partial hierarchies.

4.4 The task model tool

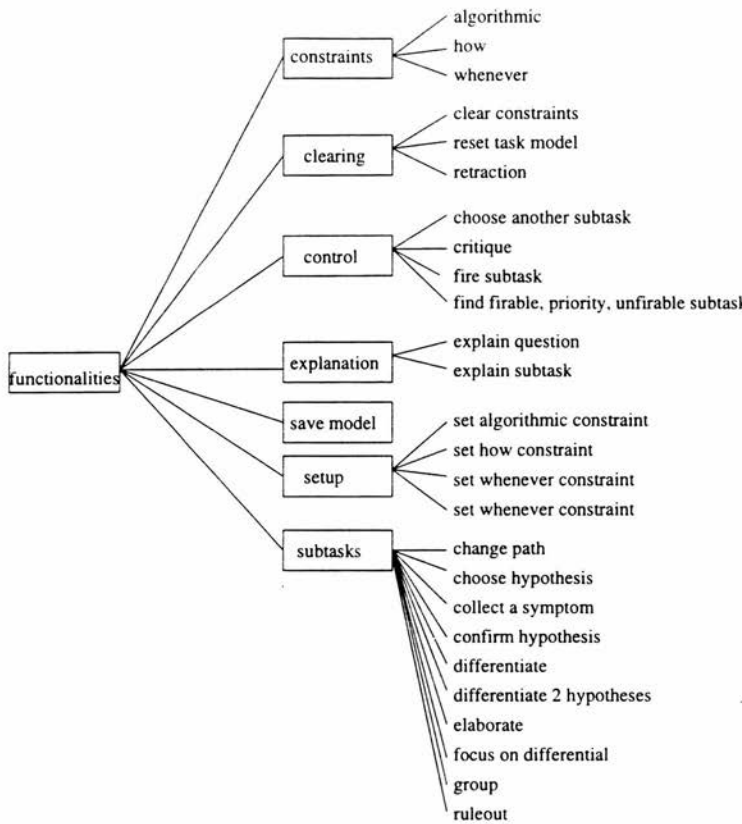


Figure 4.4: Structure of the task model tool

The task model tool (figure 4.4) consists of a set of diagnostic subtasks, facilities for choosing and setting three types of constraints on subtasks and for testing and critiquing the model thus built on different case data and/or different domain models.

4.4.1 Subtasks

In Chapter 2 we briefly mentioned some of the subtasks included in various systems which could be said to be modelling medical task knowledge, such as NEOMYCIN ([Clancey & Letsinger 84]), GUIDON-MANAGE ([Rodolitz & Clancey 89]), DEMER-EST ([Alpay 90]) and the KADS ([Wielinga *et al.* 92b]) and Generic Tasks ([Chandrasekaran 88]) methodologies. Some of the subtasks implemented here bear a resemblance to some of the subtasks in these systems. Subtasks which matched the following criteria were included either in a very similar form or altered slightly to make them more comprehensible or more amenable to use with the other subtasks in this toolkit.

- The description of the subtask must be easily intelligible to the system builder. Some of, for example, the NEOMYCIN tasks are obscure both in name and in function. The subtasks implemented here were all comprehensible to the evaluators, and they could try them out to see what their behaviour was like.
- All subtasks must be mutually exclusive. In some systems two or more subtasks may have similar effects and be distinguishable really in name alone.
- The subtask must be identifiable as a real bit of diagnosis, and not just be computationally convenient.
- The subtask must not rely on domain specific or dependent data. For example, the NEOMYCIN task 'process-hard-data' violates this criterion. Using soft data before hard data is a 'cost effectiveness' heuristic which could be implemented using a 'how' constraint in our system but should not be part of the inference procedure. This not only enforces the domain/task distinction in the models but also avoids prejudicing the builder's choice.

Structure of a subtask

In the current task modelling tool subtasks are implemented in an object system which allows for the addition of slots. All subtasks have the following slots:

- A **method** slot for firing the subtask: This is what calls the subtask's procedure from an external file. This is to ensure that the system is not entirely dependent on the KEE object environment.

- **Algorithmic effects**: This is how the ordering constraints act when the task model is operating. When a subtask is fired its algorithmic effects automatically operate. Algorithmic constraints which are effectively ordering relationships between two subtasks cause flags such as 'now' and 'never' to appear in the whenever constraints of the second subtask in the relationship. A one-place algorithmic constraint like 'once-only' will cause a flag to appear in the subtask's *own* whenever constraint. These are picked up by the control mechanism which looks first at subtasks which are thus labelled (urgent and rejected subtasks).

- **Changes**: Which diagnostic objects can be changed by the firing of the subtask. This information is not currently used but is potentially available for more sophisticated explanation facilities.

- **Explanation**: Contains a very short description of the subtask procedure. This is accessed by a simple explanation mechanism available to the enduser.

- **Whenever constraints**: a list of state descriptions concerning diagnostic objects which must be met for the subtask to fire.

- **How constraint**: The name of a choosing function which will be called whenever the subtask is fired. Only relevant on subtasks that have to make some choice amongst a list of symptom or hypotheses. The default is 'first'.

Subtasks currently implemented

The following are the subtasks currently available to the system builder in the task model tool:

- **Change-path**: Chooses a general hypothesis from the focus using **choose-hyp** (see below), looks up that hypothesis' children in the domain knowledge base and

overwrites the focus with those child hypotheses that agree with the known case data.

- **Choose-hypothesis**: Chooses a hypothesis, using **choose-hyp**, from the focus and overwrites the focus with it.

- **Collect-a-symptom** This is the most basic subtask with which to enter case data. Collect-a-symptom presents the hierarchy of symptoms in the domain knowledge base from the root down as a cascading menu. At each choice-point in the hierarchy the user can choose an individual node or, if any children of that node exist, choose a more specific symptom. It is only possible to enter a symptom which exists in the domain knowledge base, by choosing, thus obviating the possibilities of misspelling or other data-entry inaccuracies. Collect-a-symptom adds the symptom to the contents of the diagnostic object Known-symptoms. It then recalculates the differential and focus. (See 'calculate-differential' below).

- **Confirm-hypothesis**: Chooses hypothesis from the focus using **choose-hyp** and tries to confirm it by asking the user about one of its symptoms, chosen by **choose-symptom** (see below). The user's answer is added to known-symptoms, if the symptom is confirmed, or false-symptoms if it is denied. The differential and focus are then recalculated on the basis of this new data.

- **Differentiate**: **Group**s the focus hypotheses and then chooses one of these hypotheses' symptoms to ask about using **choose-symptom**.

- **Differentiate-2-hypotheses**: Chooses one of the symptoms unique to either of the first two focus hypotheses using **choose-symptom**.

- **Elaborate**: Elaborate looks up all the children of each focus hypothesis in the domain model and writes these child hypotheses into the focus (providing, of course, that they are consistent with all the known and false symptoms already gathered).

- **Focus-on-differential**: Overwrites the focus with the contents of the differential.

- **Group**: This subtask is the direct opposite of the subtask 'elaborate' described above. Group looks up all the parents of each focus hypothesis in the domain model

and writes these parent hypotheses into the focus (providing, of course, that they are consistent with all the known and false symptoms already gathered).

- **Ruleout**: Uses **choose-hyp** to find which hypothesis to rule out. Finds all the symptoms of the hypothesis that don't appear in any other hypothesis in the differential. Chooses one using **choose-symptom** and asks the user about it.

- **Calculate-differential**: This is not a subtask in its own right but a crucial function used by any subtask which collects new data. In other systems using diagnostic subtasks (see Chapter 2) something like calculate-differential is implemented as a separate subtask. We have removed it from the subtask battery as every time new data is collected something like this must be done and having to ensure this calculation is done every time puts an unnecessary workload on the system builder. The procedure looks up the domain knowledge base and writes into the differential only those hypotheses that have all of the known symptoms and none of the false ones. It similarly checks the contents of the focus.

- **Choose-hyp**: Not a subtask in its own right but is used by **change-path, choose-hypothesis, confirm-hypothesis** and **ruleout**. Chooses a hypothesis from a list of hypotheses on the basis of what how-constraint has been set on choosing hypotheses. If no how-constraint has been set, the first hypothesis is chosen.

- **Choose-symptom**: Again this is not a subtask in its own right but is used by **confirm-hypothesis, differentiate, differentiate-2-hypotheses and ruleout**. It chooses a symptom of a hypothesis on the basis of what how-constraint has been set on choosing symptoms. If no how-constraint has been set, choose-symptom simply asks about the first symptom of the hypothesis that has not already been established or denied.

4.4.2 The case model

Executing any of these subtasks is likely to change the status of the knowledge of the diagnostician. For example, gathering any new data will reduce the number of diseases that could match the patient's symptoms. At the beginning of the diagnosis, before any

information has been gathered, any of all the possible hypotheses from the domain might be true. The current knowledge of the diagnostician is known as the *case model*, as it is knowledge about the diagnostic case under consideration. The case model will include data about a specific patient, but also relevant information from the domain model and about the current diagnosis. The case model consists of a number of *diagnostic objects*, the contents of which change as diagnosis proceeds and subtasks are executed. Some of them, such as the task-history, hold information about the progress of the diagnostic process itself. In this implementation we actually hold more case information than the system builder is likely to need, or be able to cope with, at any time. For this reason some of the diagnostic objects are visible throughout diagnosis and some are invisible. Visible diagnostic objects are marked with a **v** below.

- **Differential (v)** Holds all these hypotheses (diseases) that could possibly be true given the data that is known so far, i.e. whose symptoms are in known-symptoms and false-symptoms.
- **Differential-symptoms** Holds all the symptoms of all the hypotheses currently in the differential.
- **Focus (v)** That portion of the differential which is currently under scrutiny.
- **Focus-symptoms** Holds all the symptoms of all the hypotheses currently in the focus.
- **Known-symptoms (v)** All those symptoms which the user has either selected using collect-a-symptom or has responded positively to when asked if the patient has them.
- **Current-task (v)** The subtask which is currently being executed, or the last one to be executed if no subtask is being executed.
- **Task-history (v)** An ordered list of all subtasks which have fired, in order of firing.
- **False-symptoms** All those symptoms to which the user has responded negatively when asked if the patient has them.

4.4.3 Control

In order to understand the effect that different elements of a constructed task model have on the diagnostic behaviour of the system we must analyse the control mechanism in some detail. The control mechanism is not part of the task model per se but is what 'runs' the diagnosis both for the enduser and for the system builder when testing and critiquing the task model. The task model acts as a set of instructions which modify the control mechanism's behaviour.

The very high level algorithm for diagnosis, displayed diagrammatically in figure 4.5 executed by the control is as follows:

1. Assess whether the success criterion has been satisfied or there is no valid diagnosis, in which case stop.
2. If any subtask is 'urgent', execute it and reassess, otherwise
3. Choose a subtask whose constraints are all satisfied (i.e. match the current status of the case model).
4. Execute that subtask.
5. Reassess.

The choice of which subtask to execute is made by *prioritising* all the available subtasks. First, subtasks which have been *rejected* by the system builder for whatever reason are always ignored. It is also possible that a subtask may be system rejected because of the effect of a previous subtask's firing, as, for example when the system builder has specified that subtaskA 'excludes' subtaskB, so that if subtaskA has fired, subtaskB will subsequently never be chosen. Rejected subtasks are flagged in the task model by having the value 'never' in their 'whenever' constraints.

Next the control looks for *urgent* subtasks. These are ones that, given the status of the current knowledge, should be done immediately. A subtask may be urgent because, for example, we are at the beginning of a diagnosis and the system builder has specified

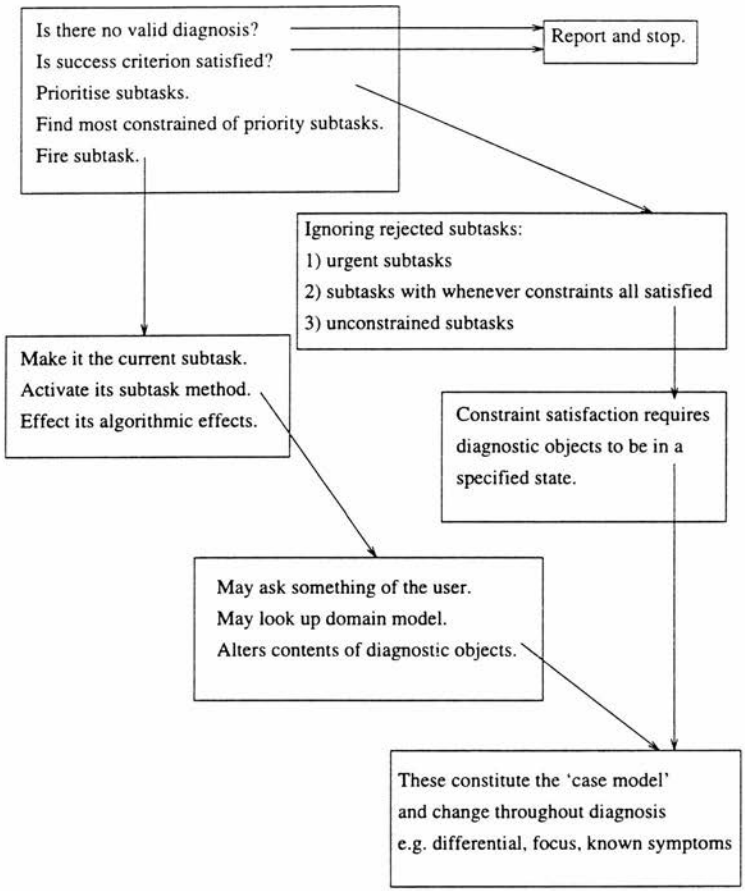


Figure 4.5: General diagnostic method

that it should come first, or because of the effect of a previous subtask's firing, as when subtaskA and subtaskB are in a 'strict-order' relationship and subtaskA has fired. Urgent subtasks are flagged by having the value 'now' in their 'whenever' constraints.

If there are no urgent tasks the control proceeds to assess for each *constrained* subtask whether its constraints are met. The constraints (whenever constraints are the only ones that matter here) take the form of a diagnostic object name and a state description, for example 'focus all-general' or 'differential empty'. The control simply applies the description (a function) to the contents of the diagnostic object. As soon as a failure is met the subtask in question is investigated no further. Of all the subtasks whose constraints are all met the control chooses the one which is most heavily constrained (or the first of these if there is more than one with equally heavy constraints).

If there are no constrained subtasks whose constraints are all met the control chooses the first unconstrained task.

In this way the control first addresses algorithmic constraints (subtasks which must or must not be done) and then considers 'whenever' constraints. A rigid task model, or those parts of the task model that are rigid, takes precedence over the more flexible model, or more flexible parts of the model.

When a subtask is chosen by the control it is fired. Firing a task has various effects. Firstly the subtask is made the *current* subtask. Current-task is a diagnostic object. The subtask's procedure is executed by activating a *method* attached to the subtask object which calls a function from an external file. The subtask's function may ask something of the enduser, gather information from the domain model's structure or the contents of slots on domain model objects, or both. In any case it will probably alter the contents of one or more diagnostic objects, as well as current-task and task-history. Many subtasks will have 'algorithmic effects', often as a result of algorithmic constraints having been set by the system builder. For example, if subtaskA and subtaskB are in a 'strict-order' relationship, one of the algorithmic effects of firing subtaskA will be to make subtaskB and 'urgent' task, in order that it be executed immediately after subtaskA. Firing the subtask activates all its algorithmic effects. The task history is updated by having the

subtask added to the end of it. The task history is also a diagnostic object.

Rather than simply having one algorithm for diagnosis, the system builder can now customise how the system they build behaves dynamically in a diagnostic situation by putting constraints on how the control mechanism operates. The control mechanism is what decides what subtask to do next.

4.4.4 Constraints

The mechanism for setting constraints in the task model tool is the principle method for the system builder to exercise control over the system's behaviour. The following constraints have been implemented in TOMKAT:

Task execution control: 'how' constraints

HOW constraints are primarily concerned with subtasks which have to make a choice. The setting of a 'how' constraint affects either the function **choose-symptom**, which is used as part of several subtasks, or the subtask **choose-hypothesis**. At the moment the system builder can only set such choices to be made on the basis of *pathognomicity* (a pathognomonic symptom is one which is indicative of only one disease, so the most pathognomonic symptom in a list is that which is a symptom of the fewest diseases) or *ordering* (they can choose the first, last or whatever).

Although few 'how' constraint options are currently implemented, this constraint mechanism provides the route for the exploitation of much domain information, particularly with regard to object relations (see section 4.3.2 above).

Ordering control: 'algorithmic' constraints

Every subtask has a slot called **algorithmic-constraints** which set ordering relations between subtasks. Ordering constraints implemented are not an exhaustive set of all possible ordering statements, but merely a token list to test the utility of the method. The following 'algorithmic' constraints are available:

- **Excludes:** Firing subtaskA causes subtaskB to be flagged as NEVER.
- **First-task:** Flag the subtask as NOW and remove this when it fires.
- **Immediately-excludes:** Puts in subtaskB's whenever constraints a condition that subtaskA was not the last subtask.
- **No-task-follows-itself:** Make all subtasks non-repeating.
- **Non-repeating:** Puts in the subtask's whenever constraints a condition that it itself was not the last subtask.
- **Only-once:** Firing the subtask causes it to be flagged as NEVER.
- **Order:** adds to subtaskB's whenever constraints the condition that subtaskA must be in the task history.
- **Reject:** Flags the subtask as 'never'.
- **Strict-order:** Firing subtaskA flags subtaskB as NOW.

State analysis: 'whenever' constraints

The types of 'whenever' constraints available represent states that diagnostic objects can be in. Again this is not an exhaustive set of all possible state descriptors on diagnostic objects but merely a token list of states to allow evaluators an idea of the range of possible constraints a system builder might like to set:

- **all-general:** There are no leaf nodes from the domain model.
- **empty:** There are no members.
- **getting-big:** Currently set at having more than five members.
- **none-general:** Contains only leaf nodes from the domain model.
- **not-empty:** There is at least one member.
- **one-left:** There is only one member.

- **two-left:** There are only two members.

Extensions to this list suggested by the evaluators are discussed in Chapter 6.

Criterion of success

The criterion of success is for telling the control when to stop. It is implemented, and set, in exactly the same way as a whenever constraint and indeed can be set on any diagnostic object, although it may only really make sense on some of them. Some evaluators suggested that as well as gauging success by say, the size of the contents of the differential (like one-left) a system builder might also want to determine success by how long the task-history was or how many symptoms had been gathered. These enhancements are discussed in Chapter 6.

4.4.5 Testing the model

The task model tool incorporates several mechanisms whereby the system builder can observe the behaviour of their task model in different diagnostic situations. The effect of individual subtasks can be explored, interactive diagnoses can be 'stepped' through and the choices exercised by the control mechanism can be examined in detail. Facilities available are:

- **diagnose:** Runs the control algorithm until success or failure.
- **explain:** Shows a brief description of the current subtask.
- **reset:** Starts another consultation with the same task and domain models.
- **retract-symptom:** For correcting erroneous data entry. Both positive or negative responses can be retracted.
- **select-dkb:** Allows use of another domain model. Several samples are included and the system builder can create more than one domain model.
- **select-task:** To choose a particular subtask to fire.

- **show-firable-tasks:** Displays those subtasks which could fire given the current state of knowledge.
- **show-priority-tasks:** Displays those subtasks have highest priority to fire given the current state of knowledge.
- **show-unfirable-tasks:** Displays those subtasks which cannot fire given the current state of knowledge.
- **step-control:** Runs the control algorithm one subtask at a time.

The most important part of these testing facilities is the *critiquing* mechanism which ensures that the system builder can control how their task model will perform for the enduser. It allows them to systematically alter the model until it behaves exactly as they would wish. TOMKAT also gives the system builder an option to experiment with a default task model. Under this critiquing method the system controller uses the default task model to step through a diagnosis accepting or rejecting its behaviour at every step point. If a behaviour is rejected, the system builder can choose another behaviour and give reasons for this choice, or simply give reasons for the rejection. Critiquing proceeds thus:

1. Controller chooses a subtask on the basis of current knowledge in diagnostic objects and set constraints.
2. System Builder accepts it, controller fires subtask and updates diagnostic objects or
3. System Builder rejects it and
   (a) chooses another subtask (including 'any other') and gives a reason for the choice in terms of a new constraint or
   (b) doesn't choose another subtask and gives the reason for rejection as a new constraint.
4. Controller chooses a subtask on the basis of updated model.

This process can be repeated indefinitely on multiple diagnoses, using different domains if required, until the system builder is satisfied with the model's behaviour.

The representation of a task model simply as a set of constraints on subtasks is very abstract and remote from the eventual behaviour of the model in a real problem solving situation. Responsive task models in particular can be difficult to grasp, and building such a task model can seem remote from the real thing. This method of critiquing the behaviour of a default task model or a task model that has been partially constructed by the system builder allows them some idea of the consequences of their actions; the effect of choosing and ordering subtasks and setting conditions on their execution.

4.5 The users' tool

Once both domain and task models have been constructed to the system builder's satisfaction they can be made available as a complete system. This will perform diagnosis under the control of the control mechanism, guided by the instructions in the task model. When subtasks are fired they may look up information in the domain model or ask for case data from the end user. The facilities available to the end user are a reduced set of what is in the task model tool, namely **diagnose, explain, reset, retract-symptom, select-dkb, select-tasks, show-firable-tasks, show-priority-tasks, show-unfirable-tasks** and **step-control**.

This is a large set of facilities for potential users, in fact probably over large. It is likely that such a set would be unnecessary or even confusing for users who are wanting to use the models in a pure 'consultation' mode. We did not test the systems built by evaluators on real end users (see Chapter 5) as we were concerned mainly with the performance of TOMKAT as a tool for *exploring and building* models. Such testing would undoubtedly reduce the set to an appropriate and manageable group of facilities.

4.6 Summary

We have here described a toolkit implementing the four major aspects of the methodology outlined in Chapter 3; the separation of domain and task knowledge, the modelling of domain and task knowledge, building models from subtasks and building responsive task models. The toolkit consists of two tools: a domain model tool for constructing an object hierarchy of diseases and symptoms, and a task model tool for controlling diagnostic behaviour in the execution of subtasks. The domain model tool features facilities to create and maintain the object hierarchy and relations between those objects, incorporating a wide scope library object hierarchy from which useful elements can be borrowed. The task model tool allows the specification of rigid, responsive and mixed models with the use of three types of constraints on the execution of subtasks at runtime of the task model. These constraints control how the subtask is to be executed ('how' constraints), the ordering of subtasks ('algorithmic' constraints) and preconditions which must obtain before the subtask can fire ('whenever' constraints). 'Algorithmic' constraints enable the construction of rigid task models and 'whenever' constraints allow more flexibility. Facilities are included for testing the behaviour of task models under different diagnostic conditions, and for critiquing and adjusting these models to ensure appropriate diagnostic behaviour. Once both domain and task models have been built, they combine at runtime in the end user's tools. Diagnosis proceeds according to the instructions from the task model to the control mechanism. The current knowledge of the system is kept in a case model consisting of various diagnostic objects. These hold both the responses from the enduser and relevant information which subtasks gather from the domain model.

Chapter 5

Evaluation

In this chapter we assess the two major elements of this project, the *concept*, the set of ideas outlined in the methodology of Chapter 3, and the *product*, an implementation of that set of ideas as the TOMKAT system described in Chapter 4. It is important that both be evaluated, and in particular that the concept be evaluated independently of the product to avoid imperfections of implementation obscuring any intrinsic value of the concept itself. This evaluation package is thus geared towards assessing both how appropriate and comprehensible the concept is to a varied group of potential users and how usable the product is by those same individuals.

In Chapter 1 we stated that the aims of this project were

> "to develop a realistic method of making diagnostic strategy explicit and to produce a knowledge acquisition tool implementing that method which allows the analysis and synthesis of diagnostic strategies so that learners can explore such strategies within a real framework and domain experts can create and customise them to be incorporated flexibly into diagnostic support systems."

The aim of this evaluation is to assess how far we have achieved these goals by asking two major questions:

1. Can people understand and relate to the methodology?
2. Can they use the system to explore and build models of diagnosis?

5.1 Method

Because we have continually stressed the importance of the experts' direct involvement in the knowledge acquisition process in this domain, this evaluation is very much geared towards assessing the adequacy of both concept and product from the user's viewpoint. The practical evaluation was carried out with eight subjects with a range of appropriate experience. It consisted of a structured interview exploring the concept as related to the interviewees' experience and a structured task in which subjects performed standard exercises of increasing complexity using the product. The interview took approximately 25 minutes and the structured task about 40 minutes for each subject. All interactions were tape recorded with the agreement of the subjects and transcribed verbatim. Full transcripts are not included in this thesis because of their excessive length but short quotations from the transcripts are included where appropriate.

The topics covered in the interview were divided into the categories outlined in section 5.1.2. All comments in the transcripts were then assigned to one of these categories. The comments were then summarised in the results section and, where many subjects had a similar response, or individuals had a particularly interesting response, these are reported. We have also included in this section any comments which the subjects made on the textual description of the product which they were given prior to undertaking the structured task. For the structured task we observed and noted the subjects' capabilities on each part of the task. This was also tape recorded and transcribed. The topics covered in the structured task were as described in section 5.1.3. All comments and notes on performance were assigned to one of these topics and, as before, the corpus was generalised. In both concept and product evaluation there is also a small section labelled 'Misunderstandings'. Misunderstandings are often very revealing not only about the way the subject has interpreted what they are evaluating but also about hidden ambiguities, misleading names or simply bugs in what is being assessed.

This is essentially a qualitative evaluation which is semi-formal in that pre-set topics and questions were covered in unrestricted interviews. Subjects were allowed to answer questions freely and, in many cases, a topic was pursued which was not anticipated in

the initial specification. In particular, it emerged that those with medical training were keen to discuss the lack of emphasis on diagnostic method and strategies in that training and were interested in their own diagnostic strategies as well as what might characterise those of other specialities. We have reported on this data in a separate section 'The Art of Diagnosis' below (section 5.2.3) as we feel that such opinions, far from being tangential, are germane to this project. Similarly, several subjects spontaneously raised the interesting issue of representing and validating task models, which was not anticipated in the interview. This is included in section 5.2.1.

We have not performed formal quantitative evaluation. Such an evaluation would be appropriate to the trial of a system prototype but would not be able to gather the sort of anecdotal information which has proved particularly valuable especially in the concept evaluation. The authors of the GUIDON-MANAGE system [Rodolitz & Clancey 89] which we reviewed in Chapter 2 as closely related to the TOMKAT system also eschewed formal data analysis because

1. Their subject group was too small for statistically significant findings (n=5 for them)
2. The subjects varied too much in prior experience with diagnosis.
3. The subjects were just becoming familiar with system by the end of session.
4. The system itself varied from session to session.

These observations, with the exception of the last, also hold true for this evaluation.

The evaluation is formative rather than summative. The toolkit described in Chapter 4 is a prototype which is designed to test out the practicability of an idea rather than something domain experts could be expected to use immediately. In the evaluation the implementation is tested by a range of subjects to expose the modifications necessary to both the concept and the product to enable it to be used by experts in the medical domain with little or no AI training. The potential improvements to the system exposed by this evaluation are outlined in Chapter 6.

As our main focus of interest was on task modelling we evaluated the task modelling tool in the product evaluation to the exclusion of the domain modelling tool. Similarly, we did not test the usability of the models our evaluators produced because

1. the target endusers of these models would in reality be the choice of the experts themselves and so
2. if we were to test the usability of the models built by the experts we would not be testing our toolkit but rather their ability to build appropriate models for their chosen endusers.

5.1.1 Subjects

The subjects who performed the evaluation came from as wide a range of experience in the area as possible. There are two types of expertise relevant to evaluating this type of project; experience in AI and experience in medicine. We were also interested in the opinions and performance of individuals whose expertise spanned both of these areas, who are much harder to come by. This wide, shallow spread of experience was to avoid potential problems of

- those knowledgeable in AI being overly influenced by their modelling experience so that they would not be able to see the drawbacks of applying the theory in a practical domain and
- those with medical experience being too distracted by the technical aspects of the application to see its practical implications.

In the end these fears were proved largely, although not completely, groundless. It was helpful to be aware of an evaluator's background when assessing their responses. For example, the doctors all stressed their experience as students of having a rigid history-taking method 'ground in' so much that they felt they asked questions without thinking. This obviously affected the type of models they initially wanted to build and also the tendency to be question oriented rather than goal or task oriented in their diagnostic strategy. The AI experienced people on the other hand almost took the modelling

perspective for granted as a Good Thing, although they were very aware of potential difficulties which users without a 'knowledge engineering' background might have.

It was important to have some evaluators with medical experience and limited, or no, computing or AI experience as we wanted to assess the potential for this type of tool to be used by real experts in a domain rather than by knowledge engineers. The inclusion of evaluators with an AI background, especially in modelling, was to gather informed opinion on the appropriateness of the ideas expressed in the methodology as well as the technical adequacy of the implementation. We were also anxious to include some subjects from the medical side who were involved in the training of medical students as we have previously emphasised the pedagogic advantages of the system and methodology. We were unfortunately unable to perform any evaluations with individuals with medical experience in General Practice. As emerged from the interviews with those involved in hospital medicine (see section 5.2.3), this would probably be much more apposite experience for the type of system we are considering, as many of those involved in hospital medicine felt that diagnosis per se was done rather rarely in such a setting. It would also have been advantageous to include some medical subjects at an earlier stage of training for the same reasons that we wished to include trainers. Attempts to procure medical trainees, and GP trainees particularly, were unsuccessful.

The demography of the final subject group (n=8) was as follows:

- Four subjects had an AI background but little or no medical knowledge.
- One subject had AI and medical knowledge.
- Three subjects had medical but little or no computing or AI knowledge.
- Medical experience ranged from senior house officer to senior registrar and included two subjects with training experience.

There was no control made for age, sex or other factors as the sample size was small and it seemed unlikely that such considerations would affect the subjects' responses.

5.1.2 Concept evaluation: Testing the methodology

Concept evaluation was conducted by structured interview based around a short introductory text which described in simple terms the ideas explored in the methodology of Chapter 3. Evaluators were asked to read this critically at their leisure. The interview was then recorded. The interview protocol and introductory text are included in full in Appendix B. The main topics covered in the interview and subsequently used to categorise the reviewers' comments correspond to the elements of design methodology outlined in Chapter 3, although we have separated out and concentrated on some of the more practical implications, such as the activity of diagnosis using subtasks and the use of constraints when building task models. As we did not go on to evaluate the domain model tool in the next exercise the introductory text and interview questions are focussed on elements of task modelling, and especially those unique to this project. The topics covered were as follows:

1. The division between task and domain knowledge
2. Doing diagnosis with subtasks
3. Building models of diagnosis with subtasks
4. Rigid and flexible models
5. Constraints
6. Representing and validating the model

The last topic was not initially part of the interview but arose spontaneously from several subjects' thoughts on the types of task models that people might build. The interview also included a 'general' section which asked only subjects with a medical background about their experience with practical diagnosis. The volume and quality of the responses to these questions resulted in the 'Art of diagnosis' section (5.2.3) below. For each of the topics we attempted to ascertain

1. Can they understand the idea?

2. Is it an appealing idea?

3. Is it anything like what happens in practice?

4. If not, is it a potentially useful idea?

5.1.3 Product evaluation: Testing the tool

The product, the tool described in Chapter 4, was assessed by asking the subjects to perform a structured task using the tool. This evaluation was confined to the task modelling tool as we felt that the limited time available with each evaluator was best spent on the most theoretically interesting part of the project. Evaluators again read an introductory text outlining the functionality of the toolkit (in Appendix B). They were then asked if there was anything in the text which required further explanation. The responses to this were assimilated with those of the concept interview. The purpose of the short product interview was to give the subjects a broad brush introduction to the system so that they might know what to expect when they first confronted the machine. The topics covered in the introductory text were:

1. The subtasks

2. Rigid models: 'algorithmic' constraints

3. Responsive models: 'whenever' constraints

4. Critiquing

The subjects were then asked to perform a standard structured task using the task model tool on pre-prepared domain models. The structured task was, with a simple domain knowledge base which covers childhood diseases to perform the following exercises:

1. **Interactive diagnosis:** Do diagnosis by interactively choosing subtasks.
   Purpose: This was to allow subjects to explore the effects of the various subtasks and see if they correspond to anything they do in practice.

2. **Building a rigid model:** Look at the contents of the task history which retains the steps of the interactive diagnosis and build a rigid model based on this record.
   Purpose: This gets across practically the idea of building a model of diagnosis independent of the actual diagnosis and using algorithmic constraints.

3. **Building a flexible model:** Test the rigid model on a different example, hopefully it will fail. Using the critiquing method, turn it into a more flexible model using whenever-constraints.
   Purpose: This enables subjects to explore constraints on subtasks to see which are plausible and how they affect the operation of subtasks.

4. **Using the model on other examples:** Run the model on different examples within the same domain and on other test domains available in the toolkit.
   Purpose: This shows how the flexible model responds to different input and is still able to do diagnosis adequately

although it was not expected that all subjects would get though this complete list. When observing the subjects' performance on these exercises we tried to assess:

1. Do they understand the implementation?
2. Can they use it?
3. How would they improve it?
4. Does using it help to understand the underlying ideas?
5. Did anything not work?

5.1.4 Expectations

The main expectation was that the underlying notions on which the toolkit is based would be progressively more difficult to understand, more difficult to use and remoter from actual diagnostic practice, from the separation of domain and task knowledge through to the building of responsive task models. Because of the unfamiliarity of the concepts,

it seemed likely that the subjects would have to go through some learning process before they would be able to use the toolkit effectively. The concept evaluation phase was intended to have this effect as well as permitting prior feedback about the feasibility of the whole idea.

In the concept evaluation phase we anticipated that although subjects would recognise the separation of domain and task knowledge, they might be unable to give domain dependent examples. As regards the idea of building a task model with subtasks we would expect our medical subjects not to consciously structure their diagnoses in this way. We would anticipate that those with an AI background would not find these concepts difficult to understand.

In the product evaluation phase we would expect that in the interactive diagnosis there should be no major problems unless the operations of individual subtasks prove difficult to understand. We might expect that our medical subjects want to perform diagnosis by asking specific questions rather than explicitly choosing subtasks because of the 'unconscious' nature of their diagnostic strategies (see Chapter1 section 1.1.3). Building a rigid model should be a manageable exercise but building a flexible model may make too heavy demands on subjects who have never before been exposed to the idea of a control mechanism which can be affected by the use of constraints.

5.2 Results

5.2.1 The concept

The division between task and domain knowledge

Contrary to our expectations, all subjects were not only aware of the distinction between the two types of knowledge, but also able to give concrete examples. For our subjects the division between the two types of knowledge is very real, and is also recognised in other domains and skills. In medicine it is especially important for learners who have to make a big jump from book work where only facts are important (domain) to practice where the facts become subordinate to 'knowing how to do it' (task). Both are necessary

because as one evaluator pointed out

> "a medical student with only factual knowledge is not going to be much good at diagnosis. Task knowledge helps prioritise and order and dismiss inappropriate domain knowledge"

whilst on the other hand it is important to ensure that there is adequate domain knowledge to back up the task knowledge.

Although the division is real it may be fuzzy and difficult to make judgements about which category an individual 'piece of knowledge' falls into. There may also be other possible levels and divisions such as

> "defining what you want to achieve which in medicine is always defined anyway as you want to achieve an explanation of the symptoms".

This is probably more true with tasks other than diagnosis, and the distinction may also be more difficult to make with synthetic tasks like design or planning because these tasks themselves are less well understood.

Doing diagnosis with subtasks

All subjects thought that the method of doing diagnosis by subtasks made sense, although one AI respondent thought that an ordinary user of the system might not naturally be able to do it. Several from both backgrounds observed that although diagnosticians might use this method in practice it would not be explicit but subconscious, as we anticipated in section 5.1.4 above.

> "You do that in the way you come to diagnoses subconsciously. You don't obviously go through saying this is the next subtask but that would be the way you go about it"

and also

> "I don't think you consciously say I'm going to group these into y'know if you think there's an infection somewhere in the body then you think in terms of infections."

The individual subtasks were felt by both groups to be acceptable, obvious, the sort of subtasks one would expect to find as elements of diagnosis and at the right 'level of granularity' (one of the AI subjects complained that KADS' subtasks (see Chapter 2 section 2.2.2) were at too fine a level of detail). One medical subject mentioned that in order to understand individual subtasks she needed to imagine in what situations they would be used, thus anticipating the 'whenever' constraint setting in the implementation. Another medical subject pointed out that grouping is done a lot in medical practice, especially by bodily systems e.g. the symptom of chest pain could make a diagnostician think of perhaps cardiac, pulmonary, circulatory and digestive systems. An AI subject pointed out that differentiate is also the main subtask used in troubleshooting, a subtype of diagnosis.

Building models of diagnosis with subtasks

All subjects appeared to understand the idea of building a task model of diagnosis with subtasks. It was generally recognised that there is not just one model of diagnosis, one AI respondent pointing out that

> "It's always possible to break a task up into identifiable bits it's how you structure these bits together that is the crucial component"

and another noting that the order in which subtasks are used will vary widely with experience. A medical subject observed that an efficient computational diagnosis could be done by simply entering all the symptoms available and calculating which diseases matched but this would not be helpful to human users. She felt that using the methodology of this study would be 'more like what humans do', thus approaching the desirable quality of system which can be *seen to be doing diagnosis*, in the way that humans perform this task. Again several AI subjects drew attention to the difference between synthesis

and analysis tasks, suggesting that building models out of subtasks works fine for the latter but that synthesis tasks might be more difficult, again because they are less well understood.

All the AI subjects were concerned about the difficulties in acquiring the task knowledge, both getting hold of the building blocks and getting the experts to build models with them. They felt that it was a very valuable exercise to identify the subtasks, particularly as some might be have wider applicability than diagnosis, being true generic problem solving units. Again the special status of diagnosis was noted and that

> "it's easier to acquire the task structures doing something like diagnosis because we know quite a lot of the task structures of doing diagnosis so we know what sort of elements or fragments to be on the lookout for the sorts of behaviour that would be regarded as task structures"

It was suggested that in order to get useful models the experts would need examples of complete models "You couldn't just present them with the bits and say 'produce one of these'."

Rigid and flexible models

As anticipated, the distinction between rigid and responsive models of diagnosis was more difficult to grasp for everyone. Several subjects expressed difficulty in getting to grips with the distinction until they actually saw examples in the implementation. One medical subject felt that he could not work with the flexible model, although it was a valid method, and that he always used the same rigid model. This was mirrored by an AI subject who observed that in troubleshooting, where the task is a lot less complex than diagnosis, people always use the same rigid model and nobody would want to do it otherwise.

Several subjects (5) suggested that the procedural method is the initial way of learning many problem solving (and other) skills, and that this is the way many tasks are taught, but the responsive method is more characteristic of experienced experts' behaviour who

"learn to be adaptive". Tasks are probably taught procedurally because it is much easier to describe them this way and thus pass them on, but the responsive method is more useful, more powerful, more efficient, more general and better when things go wrong in a real world problem solving situation where it has to respond to vagaries of input data and recover well. Two medical respondents also pointed out that the responsive method is more 'human' in that it "gives some reasons for doing things", one also suggesting that it could be "responsive to the local need".

Several respondents (both backgrounds) mentioned the difficulty of getting hold of the responsive model, particularly as it is difficult for the expert to assess what they are doing, to

- "step into your head and find out what the process is because you stop consciously thinking about things".

This unconscious nature of diagnosis and hence its inaccessibility (see Chapter 1 section 1.1.3) is one of the main hurdles for task knowledge acquisition. One solution suggested was to get at the responsive model by actually posing a diagnostic problem. This is in fact the way TOMKAT works in its 'critiquing' mode (see Chapter 4 section 4.4.5) where the system builder can investigate the behaviour of the constituents of the task model on real domain knowledge bases.

Constraints

It was, predictably, the AI subjects who had most to say about constraints. The medical subjects all expressed some difficulty in digesting the idea and thought they would understand it better in the implementation. The explanation in the introductory text was lacking here, as evidenced by all the medical subjects asking for clarification on this part of the document.

The method of model building using constraints was favourably received, one AI subject pointing out that although the method is time consuming the gain is that once a successful model has been built it can be used on other and simpler diagnostic problems.

A serious negative observation was that describing the sequencing of tasks in terms of constraints (as with the rigid model) is potentially confusing and complex. Another respondent said that they were not quite confident about how to combine the two types of constraints, and this is an aspect of the same problem.

Comments about constraints also centred on how to get people to use them to build models. As with subtasks, there is concern first about stocking the 'library' of constraints, the bases on which people might choose to execute subtasks, and second about people's ability to recognise them and use them to build models. It was generally felt that it would be unwise to simply present constraints to users and expect them to build models and a method proposed for introducing constraints:

> "It's simpler to present first of all the concept of a sequence of tasks and then to say it doesn't need to follow the exact sequence to introduce the concept of constraints rather than to say up front you just set up some constraints which say this"

Two AI respondents mentioned that it is difficult to have anything other than an ad-hoc set and that, given this, it would be advantageous to have facilities for adding new constraints or altering existing ones. Other extensions to the constraint mechanism were suggested. First was allowing whenever constraints to say things about the state of diagnostic objects such as the task history, the subject pointing out that this would in effect blur the distinction between the two types of constraint. In fact this is possible in the current version of TOMKAT, although we did not explore the implications of setting such constraints. Second was the desire to specify how particular subtasks would be executed, anticipating the 'how' constraints which were not included in the evaluation. Two subjects independently made this suggestion, one wanting to differentiate using a question that would give the most information, suggesting that 'a binary chop' of the candidates is the most efficient. The other wanted some control over which questions are asked with regard to variability such as that particular tests might not be available or be too expensive.

Representing and validating the model

When discussing constraints and the two types of models several subjects anticipated critiquing as a method of building models, noting that people just do not have enough time or confidence to build models from scratch.

> "They need to see examples of whole ones and perhaps maybe arrive at a particular one by refining those by playing around with various subtasks and then doing extra things ..... changing the constraints on subtasks."

One AI subject also said that this is the method they use in practice, presenting a task structure to the informant, who then critiques it.

Many respondents spontaneously raised the issue of how difficult it is to represent a task model. This generally emerged when subjects were performing the structured task but several alluded to it even before they had seen the implementation. The general problem is that building a model of a task is a very abstract process, producing an abstract product. The only way that the system builder can get a feel for the model is to observe its behaviour. As far as the two types of model go, the flexible model is even more abstract than the rigid one because it in a sense does not exist until a diagnosis is actually undertaken. The model is almost created 'on the fly' as a response to the available data. This means that it is very difficult to represent. Where the rigid model could at least be represented by an ordered list of subtasks, the flexible model would be much more like a piece of programming code involving contingency statements about the status of case knowledge. One subject suggested that since

> "there may be a conflict or redundancy between algorithmic constraints on a subtask it would be nice to see visually whether you have created a conflict."

Many subjects expressed the desire for some sort of graphical representation, although acknowledged that this would be very difficult to implement as it is not clear what the representation should be. There was a sense that a graphical tool would be "something that would make sense of this jumble of unordered constraints". It was felt that a

graphical presentation could give a better idea of the action of subtasks as well, especially if it involved some presentation of the subtask working through the domain

> "if you could see visually what's happening in terms of the taxonomy or classes of diseases and symptoms"

The feasibility of these suggestions will be discussed in Chapter 6.

Misunderstandings

Several misunderstandings emerged which revealed interesting aspects of the presentation of material, as well as about the subjects' own state of knowledge.

Two misunderstandings arose from the naming of entities, the first that whenever constraints are demons, procedures which are done automatically *whenever* a situation is seen and the second that the 'elaborate' subtask involves asking more questions about something.

One medical subject was a little confused about the two types of models

> "The algorithmic way of going about it is very much the medical school way of doing it but the responsive model is like differential diagnosis."

This is half correct in that the 'medical school' model is definitely rigid, but the misunderstanding lies in saying that differential diagnosis is a responsive model.

The last interpretation rather than misunderstanding is interesting because it came from someone who was very concerned about the value of diagnostic supports in teaching

> "That [responsive] seems it's being built for people with less knowledge as it's virtually making a diagnosis for you."

This subject was concentrating mainly on the end product of the model building process, the system that the enduser gets, rather than the model building tool itself.

5.2.2 The product

A general point needs to be made about this product evaluation. As we were testing the task modelling tool only, we operated with quite small domain models, although they were based on real domain knowledge (e.g. [Jaffa 88]). The simple domain models implemented in TOMKAT were not large or complex enough to seriously test the task modelling system and occasional erratic behaviour of a task model was generally due to the inadequacies of the test domain rather than the task model itself.

Interactive diagnosis

The most common behaviour observed in the interactive diagnosis was the initial desire across all types of subjects to keep entering symptoms using the collect-a-symptom subtask. Subjects seemed to have the idea that the system would simply go off and do the calculation if given enough symptoms. This is true, but they were missing out on the potential to have a 'cleverer' sort of diagnosis. Collecting a symptom can be done at any point but as a subtask it does no real diagnostic work at all, the user has to do the work. Reasons for wanting to put in another symptom were that

> "I feel I probably don't have enough information."
> "I feel that producing a differentiating symptom when there's one two three four five possible diseases, would help"
> "I just chose something. I felt I needed to be doing something to get closer to the solution".

Subjects did not seem to realise that the system would help them to choose what information to put in next if they approached it at a more abstract level. Although common, this behaviour was only observed at the beginning of diagnosis. Subjects soon got the hang of using the power of subtasks to actually go and do something computational rather than just pumping in information.

Most subjects then approached the problem as a matter of continually refining the focus, i.e. choosing subtasks which would reduce the list of items in the focus. This shows that

they were actually able to think of subtasks at a further level of abstraction as being 'a focus-reducing type subtask' and would have probably benefited from more subtask information of this type. This is further explored in Chapter 6. One subject expressed the wish to have more information about the subtasks and several mentioned that they only got a real feel for the subtasks' operation by actually using them

> "I feel that I'd need to go through a number of examples before I could be let loose to construct something myself"
> "With a feel for what the tasks are I've a clearer idea of what each one does"

Two medical subjects when using 'collect-a-symptom' wanted to enter something as specific as possible, reflecting the method of diagnostic practice described by several subjects of trying to find as much about the presenting *symptom* as possible (see section 5.2.3) because a very specific symptom will often lead to a very specific disease. Medical subjects did have a tendency, as predicted in section 5.1.4 above, to want to ask specific questions on seeing the contents of the focus and differential rather than to use a subtask. One medical subject saw an 'outlier' in the focus which she wanted to rule out because it is a dangerous disease and is easily ruled out (pyloric stenosis only occurs in infants and is characterised by projectile vomiting). This is an obvious situation where the implementation of how-constraints would be helpful, ruleout being executed on the basis of ruling out the most dangerous disease first. Another saw four diseases in the focus and immediately realised that a question about urinary symptoms would reduce this (the same question that the differentiate subtask would choose for her). It was evident that these medical subjects in a sense knew their subject too well for our purposes and were able to jump to the question that would be asked to achieve the goal of reducing the focus without considering what problem solving process they were undertaking to do it.

Most subjects seemed to use and be happiest with the behaviour and concepts of 'collect-a-symptom', 'group','ruleout', 'confirm', 'choose-hypothesis' and 'differentiate'. 'Elaborate', 'focus-on-differential' and 'change-path' were used less often.

One AI subject thought it would be useful to have the system recommend a subtask, if required, when going through an interactive diagnosis.

Building a rigid model

All the subjects found the exercise of building a rigid model based on their previous interactive diagnosis as recorded in the task history straightforward. Observations in this exercise are thus somewhat fragmentary as there were no major problems. All the subjects were confident with the mechanisms of building the rigid model and realised that its inflexibility was what caused it to fail when reusing it on different data (although one subject wanted to use the same example throughout). Several subjects discovered the inadequacies of the rigid model before actually running it to breaking point. Two subjects wanted to use 'whenever' constraints when building a rigid model. One subject wanted the equivalent of an 'until' statement, which would be part of a flexible model, and one wanted to do a subtask ('group') when there were more than ten items in the focus, again anticipating whenever constraints.

One AI subject suggested that since 'no-task-follows' itself is a very general ordering statement it might be automatically set, although there might be situations where repeating tasks would be desirable. 'No-task-follows-itself' is designed to avoid fruitless 'looping'. There is also provision to avoid cycling two tasks with the algorithmic constraints but it is more difficult to detect and avoid bigger loops. There are also situations in which cycling could be desirable, until a certain state is achieved, but the system can only put constraints on single subtasks or pairs, not groups of them.

An AI subject wanted to know on what basis choose-hypothesis makes its choice, anticipating how-constraints.

One AI subject felt that building the rigid model was like saying 'I would always do it this way'. The same subject also wanted a way of saying that confirm-hypothesis would be the last thing in the model. However there is no real way of specifying what 'last' is.

Building a flexible model

As expected, our medical subjects found this exercise much more difficult than the rest of the evaluation and needed a lot of help, whereas the AI subjects were all able to use

the constraint mechanism happily. We feel that the time constraints of the evaluation contributed largely to this. The evaluation was not really long enough to let subjects loose with the system, especially not those with little computing experience. Medical experts did not have time to become familiar enough with the individual subtasks and their effects to make very well grounded decisions about when they should be fired. It would have been useful to have the experts for two sessions, one a 'learning phase' which covered the material which we covered here and one an 'experimentation phase' where they would be observed building models without any significant guidance.

Sometimes people could not think of a good reason for doing a subtask (when using the critiquing mechanism), they just chose something in the hope that it would have an effect like reducing the focus. This points to a need for more information about the operations and effects of individual subtasks. This will be elaborated in Chapter 6.

One subject wanted a 'too small' condition on an object's contents and elaborated this to a desire to be able to specify the exact size of things. Another subject also mentioned a specific size constraint and indeed the 'getting big' description has an arbitrary size limit.

Two AI subjects separately suggested that they might want to have constraints which mentioned objects other than diagnostic objects, in particular the task-history. The reason for choosing a task or finishing the diagnosis might be something to do with the state of the task history or the current task, so that the model actually feeds on itself. In fact the current version of TOMKAT permits this type of constraint.

One medical subject was particularly interested in the success criterion, suggesting that there might be all sorts of different types of successful diagnosis, for example:

> " What if you were saying something like it's ok to stop if they've got a cold or flu because we treat them the same way and it's ok to stop if they've got what's the two things you've got there two other things that get treated much the same way."

One medical subject wanted to "collect a symptom when there is another symptom to

collect" i.e. to replicate the diagnostic situation where "you'd keep asking them for symptoms until they didn't give you any more". This is impossible to do.

An AI subject noticed that the same situation could be specified in different ways, such as doing collect-a-symptom either first (algorithmically) or when the differential is empty (with whenever constraints), although this would not give identical results.

An AI subject found it confusing that the control mechanism would choose subtasks to execute about which nothing had been said. There is a tendency, when building the model with constraints, to feel that rather than specifying what behaviour *should* occur the system builder is making more and more statements about what should *not* occur, in order to stave off unacceptable behaviour. The choice by the control mechanism of the 'wrong' subtask is often because several subtasks are equally heavily constrained. This subject suggested that it would be advantageous to change the priority of subtasks by some sort of weighting mechanism.

One AI subject felt that although it was 'fun' and interesting to use the critiquing method and try to change things as they went along they would probably do a lot of paper and pencil analysis first before using the tool. The critiquing would be very much more for fine tuning the model once it had been roughed out. Although critiquing is useful as a method of trying out the model, there is no way of exhaustively testing the model, especially the flexible one, because it is impossible to anticipate every conceivable set of case data.

Using the model on other domains

Only three subjects, all with an AI background really got onto this section due to time limits. It took over an hour of intensive work to get to this stage in the evaluation, by which time most subjects, not surprisingly, showed some signs of mental fatigue. All who did this part of the study found it particularly valuable to test the model on a different domain as it gave an insight into the model behaving in a similar way although it was dealing with entirely different material, as opposed to merely different *case* data as when testing the model with the same domain. The responsive task models built using one

domain all behaved rationally when tested on another domain.

One subject wanted to compare the algorithmic model with the flexible model on the same data. This can be done with the save-task-model facility, although the models cannot be compared in parallel. She felt that it would be useful to build a library of models to compare because "I don't know what each is buying me at the moment."

One subject wanted to know whether task models vary according to domains, although our test domains are not sufficiently complex to do a real test of this. Several medical subjects, however, did discuss this issue and it is explored in section 5.2.3.

Misunderstandings

Another misunderstanding due to a poor naming occurred when a subject thought that focus-on-differential was a task that would 'focus-on' a part of the differential using an unknown mechanism.

One medical subject was under the misapprehension that there were rules operating underneath the toolkit as an 'inference engine' and wanted to see a trace of such behaviour.

5.2.3 The Art of diagnosis

> "WHAT'S GUIDING YOU AS A DIAGNOSTICIAN:
> What's guiding me? A process.
> WHAT IS THE PROCESS:
> Aaaah! My process ha ha."

In Chapter 1 we drew attention to the tendency for learners in the field of medicine to be encouraged in thinking of diagnosis as a mysterious 'art' which cannot be analysed, but that experienced practitioners are expert at. In contrast to the ideal, if students are taught diagnosis at all it is at a rudimentary level. Here our medical respondents expressed their concern about about the transition from novice to expert diagnostician and the lack of support for this transition in their training programmes. We also asked them to think about some of the diagnostic methods they might use themselves and

these were very revealing especially from the point of view of modelling such processes. Lastly, medical respondents talked briefly about the differences between the diagnostic task in different settings.

How diagnosis is (not) taught

All subjects from a medical background said that they were never really taught diagnosis, or were taught only a standard set of rigidly ordered questions. This agrees with the findings from other studies which suggest that students are never given a real method or strategy to follow

> "Students are commonly given an outline of all data that they might collect, organised by 'social history', 'previous illness', and so on, suggesting that medical diagnosis is a process of collecting data in a fixed order. The result is that students sometimes collect information by rote, without thinking about hypotheses at all!" ([Clancey 88], p. 359)

Far from encouraging inferential thinking, our respondents suggested that it fostered a 'parrotlike' approach to diagnosis, one subject noting that

> "There are certain questions you have to ask because otherwise someone's going to shout at you because the answers aren't there on the piece of paper."

Students are taught to take a history in this long winded fashion and then are required to come up with a differential diagnosis (a list of possible hypotheses explaining the findings from the history and examination) and also an idea of how to proceed with the next investigation. However, they are not taught *how* to take these steps, and their only feedback is through trial and error

> "You come up with a list of possible diagnoses and you say what you would like to do next and they criticise you saying 'no a CT scan is far too expensive for someone with pneumonia."'

Several respondents suggested that this was not only inefficient but also demoralising. The reason for insisting that students learn a rigid history taking method is to ensure that they work systematically and do not miss potentially important evidence through overlooking a vital question as learner diagnosticians have a tendency to rush to a conclusion on very little evidence

> "They say oh I've seen a rash before and it was condition X. This patient's got a rash therefore it must be condition X. No there's no real inference."

5.2.4 How they do it

All the subjects expressed difficulty in accessing their own diagnostic processes, as we discussed in Chapter 1 section 1.1.3, as they felt that they had been very much internalised so that they performed almost automatically. Two crucial aspects of diagnostic strategy emerged however. The first is that the initial symptom(s) a patient presents with is/are very important in narrowing down the field of possible hypotheses. Getting a unique symptom can save a lot of 'computational' work later on. A patient presents with a symptom and questions are asked about that symptom in order to refine it as much as possible so that hopefully it will point to one or a few possible hypotheses.

> "If you have rash there's thousands and thousands of diagnoses. You couldn't search them all but if you had a particular kind of rash then there's only a few possibilities so I guess you want to narrow that symptom down as much as you can to weed out all the extraneous possibilities yeah make specific."

Patients fortunately do not present with just one symptom at once. One symptom might give an unmanageable number of hypotheses but two symptoms narrows things down a lot. Symptoms also come in clusters, and much of the diagnostic questioning is geared towards looking for these

> "You do things in groups of symptoms and you know the answer to one symptom leads you on it's almost like it leads on to questions about other symptoms. You're not necessarily thinking of the disease consciously as such

> as the fact that symptoms group together. I mean if you find people are breathless then you always ask if they have swollen ankles kind of thing."

The second factor was that diagnosticians tend to work with both disease and system domain hierarchies at once, in a somewhat mysterious manner.

> "Someone presents with a symptom and that identifies an organ system and you might pin that down but also there's the parallel process where you're identifying a disease type and so you've got these two things. There's organ systems and the things that can go wrong with them and out there there's all these multitudes of diagnoses so I guess there's quite a lot of interaction between those two things, you ask questions all the time to try and pin down the disease type and then how that influences a particular organ system or then direct more questions so you're going backwards and forwards"

This is one of the aspects of diagnosis which several respondents suggested was the most difficult for students to learn as it is never made specific. Textbooks and class teaching give two different ways of organising domain knowledge (by disease hierachy or by bodily systems) but students are never told how to merge the two representations. This is important in a diagnostic subtask like 'group' where the domain hierarchy being consulted determines entirely the types of hypotheses that will be considered.

5.2.5 Different strategies for different specialities

We asked about the difference, if any, between diagnosis in different subject areas of medicine, e.g. the difference between diagnosis as performed by a specialist in psychiatry as opposed to a specialist in endocrinology. An ENT specialist pointed out that in his area diagnosis would be quick because history taking and examination would be short, whereas in something like neurology history and examination could be very time consuming.

> "A lot of the stuff we deal with is really simple. It's walk in the door and the first sentence tells it all."

The main difference between the specialities seems to lie in two factors

1. The number of possible diagnoses in the field and
2. The distinguishability of these diagnoses

A speciality with few, easily distinguishable diagnoses (which our ENT subject felt his area to be) would have short, uncomplicated diagnostic strategies. A speciality with many, similar diagnoses such as rheumatology would have complex strategies, whereas an area such as psychiatry which has few diagnoses which are difficult to distinguish would have diagnostic strategies somewhere in between.

As well as the differences between specialisations, our subjects revealed the important difference between hospital and other settings, particularly with regard to standard procedures of diagnosis. All our medical subjects pointed out that most of the time specialists do not actually have to do diagnosis. When a patient gets to the stage of going into hospital they are often already 'diagnosed' and they are there for treatment, GPs having already performed the diagnostic filtering. Hospital procedures tend to be very standardised, whatever the ailment or suspected ailment of the patient. Standard tests, history-taking and examinations are always performed. On the other hand, there are a few patients who reach the hospital whose diagnosis is very difficult, and these are the few for whom the specialist has to perform 'real' diagnosis, although often this will be based on much of the data from the standard history and examinations. GPs are the ones who actually have to go through the diagnostic task regularly, and they also have severe constraints of time. It would be impossible to take a full medical history for each patient in a GP's surgery. However, GPs do have some history-taking advantage in that the patient is (usually) known to them.

> "The thing is they benefit from knowing the patient and of maybe knowing the patient through their life and having the whole background there and they really don't have to ask certain questions. Whereas a specialist is just given a few lines on a piece of paper about a new patient which reveals nothing about them."

5.3 Summary

This evaluation concentrated on assessing both the methodology outlined in Chapter 3 and its implementation in the TOMKAT system outlined in Chapter 4. Eight subjects with a range of computing and medical expertise contributed by both discussing the concept and using the product.

As far as the concept goes, the division between task and domain knowledge was recognised. Doing diagnosis with subtasks was seen as a sensible method, although medical subjects thought that it was probably subconscious, and the building of task models with subtasks was also understood and favourably received. The idea of there being two types of task models, rigid and flexible, was more difficult to get across, although those who grasped it felt that this was an accurate reflection of the difference between novice and expert diagnostician. The constraints method of building such models was most appealing to those with an AI background. Concern was expressed about the representation of the models, especially the flexible models, with the suggestion that some graphical tool might assist. The potential value of the ideas and system in an educational setting was recognised and especially emphasised by the medical experts.

When using the product itself, once subjects got the hang of doing diagnosis with subtasks they were proficient at using the method in interactive diagnosis. Building a rigid model based on the record of the interactive diagnosis proved simple, but building more flexible models using whenever constraints was more challenging to those with no computing experience. Further suggestions for improving the implementation, based on the findings of this evaluation, are given in the next chapter.

Our medical respondents were concerned about the lack of teaching of diagnostic strategy to students who are only ever given a set of rigid protocols to follow. On their own strategies, they revealed that diagnosis is very much *symptom led* as getting hold of a unique symptom can lead to a unique diagnosis. Diagnostic strategy may differ markedly in different specialities, but the real difference is between hospital diagnosis and 'field' diagnosis such as takes place in a GP's surgery.

At the beginning of this chapter we stated that the aims of the evaluation were to answer the following questions:

1. Can people understand and relate to the methodology?
2. Can they use the system to explore and build models of diagnosis?

In answer to our questions, people of a range of backgrounds understand and relate to the methodology. They can use the system to explore and build models of diagnosis, but those with a purely medical background need some support to build the sort of models which they recognise as characteristic of expert diagnosticians.

Chapter 6

Conclusions

In this chapter we explore how the TOMKAT system can be improved in light of the findings from the evaluation and can be developed into a usable tool. We assess how far we can go to overcome some of the problems and how some of the improvements might be implemented. We also look at how the project can be extended in applying the methodology to another area of problem solving. Finally we relate our findings to the original aims of the project and summarise its achievements.

6.1 Improving TOMKAT

The evaluation turned up many areas of potential improvement for the system, both through observation of evaluators interacting with it and through suggestions made by the evaluators themselves. Specific enhancements lie in the areas of subtasks, constraints, task-domain links and the representation of the task model.

To turn the prototype into a real system TOMKAT should be ported out of its original development environment to something less expensive and more widely available. Replicating the system outwith the KEE environment should be very easy as it was built with this aim in mind. Only interfacing and object manipulation facilities from KEE are used, the bulk of the system being implemented in Common LISP. An environment such as CLOS (Common LISP Object System) would provide adequate object manipulation and interface functionality for this (see [Fraser & Harrison 93] for a successful and simple

transfer of a comparable system from KEE to CLOS).

6.1.1 Subtasks

Although the subtasks met with generally favourable comment, and the evaluators were able to use them, there is definite room for improvement. In particular, one crucial subtask, 'differentiate' needs to be modified. Differentiate is really the most important subtask in the diagnostic process because, as the name suggests, it is about discriminating between potential candidate hypotheses and this is what diagnosis consists of in its most primitive form. Several different implementations of this were attempted, and the evaluation was carried out without a really satisfactory version being in place. Evaluators tended to use the 'differentiate-2-hypotheses' subtask instead, which has an easily understood mechanism of looking at the first two hypotheses in the focus and asking about a symptom unique to one of them. Although this works it is an over-simplified type of differentiation which ignores most of the information available in the case model.

What the differentiate subtask should do is look at all the hypotheses in the focus and ask about a symptom the value of which will split that group of hypotheses. The problem for the subtask arises in the way that the domain models are implemented. Hypotheses are not 'marked' for every possible symptom as this would be an excessive burden to put on the expert when building the domain model (i.e. it would involve each hypothesis object in the domain model having a separate slot corresponding to every symptom in the model with a positive or negative value). This means that there is not always an obvious way to perform the split of the focus hypotheses. Differentiate is thus not yet a coherent concept but probably hides a multitude of different *types* of differentiation, and even other possible subtasks.

Several of the diagnostic systems we explored in Chapter 2 perform a type of differentiation, not surprisingly. However, there is no full equivalent of the differentiate subtask in such systems (e.g. GUIDON-MANAGE and DEMEREST) because the choice of which hypothesis to pursue and which symptom to ask about is left up to the user. A system like GUIDON-MANAGE does *interactive* diagnosis with subtasks and no model is saved for potential endusers as with TOMKAT. If subtasks like differentiate, ruleout,

confirm allowed argument there would no basis for the saved model to make that choice as to which hypothesis to rule out or which hypothesis or symptom to ask about next. Allowing the user to determine interactively which hypothesis or symptom a subtask is to pursue somewhat defeats the object of the exercise. Diagnostic strategy is all about *how to decide which question to ask next.* Since differentiation is really at the heart of diagnosis, if we leave the choice up to the user at run time we are not capturing the most crucial element of diagnostic expertise.

So what is the basis on which experts choose to split the hypotheses in the focus? Are they identifying subgroups and differentiating between them, or identifying one subgroup and differentiating amongst its members or assuming that the whole focus is a cohesive group? Several evaluators mentioned differentiate as being an important subtask for their diagnostic strategies, and how they might like it to be done, such as asking a question which would split the focus hypotheses evenly or concentrating on the most dangerous hypothesis first. The obvious method for exploring how experts actually make these choices is to more fully implement the 'how' constraint mechanism and, as will be shown, this is important for other system improvements. These 'how' constraints could implement maxims like

- ask cheap, noninvasive questions first
- do not request labdata till last (i.e. soft data before hard data)
- do not ask for use of facilities you do not have (like labtests)
- look at dangerous diseases first
- look at common diseases first
- ask first about symptoms which point to single diseases (pathognomonic symptoms)

which are the sort of heuristics that diagnosticians often volunteer when asked 'how they do diagnosis'.

Although evaluators did not explicitly complain about this, it was obvious from their interactions with the system that individual subtasks need more and better explana-

tion. As a help to understanding the nature of individual subtasks, explanation facilities should include more information about their operations and effects and some indication of situations in which they might be used, particularly *what their effect in the current situation would be*. This would help in choosing which one to do next. It was evident when subjects were using the critiquing facility to build flexible models that they would also benefit from subtask *type* information (e.g. that the subtask would reduce the focus, or would ask a question of the user) again as an aid in choosing which one to do next. This could be implemented by using the information in the 'changes' slot on the subtasks which holds information about which diagnostic objects are/can be changed by the subtask firing.

6.1.2 Constraints

As we anticipated, the whole concept of constraints on subtasks as a mechanism for guiding the diagnostic method was somewhat foreign to our medically trained subjects. In Chapter 5 concern was expressed about describing the sequencing of subtasks, i.e. the rigid models as *constraints*, although it is difficult to see how else the two types of model could be combined under one representation. However, although these subjects baulked at the notion of constraints they seemed perfectly happy with the actual mechanism implemented for building rigid models. They looked on the algorithmic constraints more as a set of instructions (which they are, to the controller) on the order in which to do diagnosis. 'Whenever' constraints are more difficult to grasp, however. Again they are a set of instructions on how to do diagnosis, but they are not immediately executed as 'algorithmic' constraints are. To make users more comfortable with the constraint mechanism we need a better explanation of their functioning. When constraints are explained as 'under what circumstances would you do this sort of thing' they become much more manageable conceptually. Experts can then see their model as not a simple replication of a diagnostic strategy but rather a set of *behavioural guidelines*. This is particularly important from the training perspective. Building a task model becomes more like guiding a trainee in a general approach to diagnosis.

As far as the actual implementation of the constraint mechanism goes, minor problems

exist which are easily rectified. Generally, the constraint mechanism needs to be more friendly towards domain experts and this really means better interface features. Currently the syntax of setting 'algorithmic' constraints is poor and the visual feedback on constraints which have been set is not of the best. It is an easy matter to remove the extraneous syntactic baggage, such as parentheses, to make these more readable.

The method of prioritising subtasks in the current implementation is somewhat artificial. The algorithm (described in Chapter 4 section 4.4.3) prioritises thus:

1. ignore rejected subtasks
2. do the first of any 'urgent' subtasks otherwise
3. do the first of the most constrained subtasks whose constraints are all met

This algorithm will often result in there being several equally constrained subtasks which could fire, in which case it simply executes the first. *Weighting* constraints would allow the controller in this circumstance to make a choice rather than arbitrarily executing the first one. The algorithm would then be altered to fire the most heavily weighted subtask whose constraints are all met. A 'heavy' constraint would then come to act almost like a demon in that once the constraint is met the subtask which has this constraint will be virtually bound to fire.

Facilities for adding new constraints or altering existing ones were requested. Since many of the constraints are size-specific it is an easy matter to allow users to state exactly what number of members the contents of a diagnostic object should reach before a subtask be allowed to fire, such as that the focus have fewer than five objects, or the differential more than ten. Qualitative statements about diagnostic objects are more challenging. At the moment the only qualitative statements allowed concern the placing of hypotheses in the domain model hierarchy of hypotheses and symptoms; whether they are general (classes) or particular (instances). More task knowledge acquisition is needed to find out what types of requirements experts would like to place on diagnostic objects, especially qualitative ones. It is likely that such qualitative statements would necessitate further information to be stored in the domain model.

The same applies to the suggestion that there should be more possible (qualitative) success criteria, such as that the treatment of every hypothesis in the differential is the same. If a 'whenever' constraint, or success criterion, requires that every hypothesis in the differential have the same treatment, or that none of the hypotheses in the focus are highly dangerous, or any other qualitative characteristic, this information must be available in the domain model. However, it is interesting to note that this type of domain information is exactly the same sort as is likely to be required by 'how' constraints and would thus be implemented in the same way.

6.1.3 Task-domain links

Both the inadequacies in the 'differentiate' subtask and the desirability of having qualitative characteristics of whenever constraints and success criteria highlight the way forward for this type of system. The links between task and domain models need to be made more explicit. One of the 'symptom-led' aspects of diagnosis which our medical subjects mentioned they would like to see was the *clustering* of symptoms and to implement this method of diagnosis also requires domain linkage.

We initially proposed a strict division between task and domain knowledge (see Chapter 3 3.1.3), the rationale being that of *reusability*, all task models being reusable on all domains, and vice versa. However, from our experiences with evaluators it appears that this 'pure genericness' is neither desirable nor achievable. Task models do vary with domains and there is nothing shameful about this discovery. As was revealed in Chapter 5 section 5.2.3, medical experts felt task models would vary from one domain to another. They noted that some of the differences in diagnostic strategy in different specialities of medicine depended on the number of possible diagnoses in the field and the distinguishability of these diagnoses. The first of these features our task and domain modelling tools can cope with easily. As we discussed above, quantitative constraints on subtasks (regarding how many elements are in various diagnostic objects) are adequately handled. The issue of distinguishability of hypotheses, however, is closely related to our problem with the 'differentiate' subtask which is at the core of any diagnostic strategy.

The way to strengthen all these links first involves putting more information into the

domain models. The domain modelling tool itself needs little additional functionality; it can remain as a simple object hierarchy manipulator. The ability to add new relations ('slots') both connecting domain model objects and adding characteristics to individual objects already exists, although the domain modelling library needs more of these relations to choose from. Characteristics for symptom choice would include the availability, invasiveness or cost of tests, characteristics of hypotheses would include the dangerousness, prevalence or ease of treatment. The clustering of symptoms can be implemented with an 'associated-with' slot for symptoms e.g. breathlessness and swollen ankles, so the other associated symptoms are 'triggered' whenever one in the group gets mentioned.

Task models then need to capitalise on the extra information in the domain models. First we must allow constraints to include qualitative statements on the contents of diagnostic objects (which are all items from the domain model), e.g. stipulating that success counts as whenever all the hypotheses in the differential have the same value for 'therapy. We also need to improve the number of options on how to do subtasks which involve making a choice ('how' constraints). An option such as 'dangerousness' on the choice subtask 'choose-hypothesis' would make the selection by looking up the relative values of 'dangerousness' on its focus hypotheses in the domain model. The clustering would be picked up by the task model by requiring that if an executing subtask asked about a 'clustered' symptom the associated symptoms would be asked about automatically as part of the basic fire-task routine rather than control returning to top level.

All these additions require automatic checking features. Obviously the same values as are mentioned in the task model must be reflected in the domain. 'How' constraints which stipulate that we rule out dangerous diseases first require the hypotheses in the domain model to be marked for dangerousness, a success criterion stipulating that all members of the differential have the same treatment requires that all hypotheses in the domain be marked for treatment, otherwise the criteria can never be verified.

6.1.4 Representing the model

The activity of building a diagnostic task model is the setting of constraints on how and when subtasks can be fired in an active diagnosis. A typical task model built by this

process consists of a set of subtasks, some information as to how the subtasks should be ordered and descriptions of circumstances under which the subtasks might be used. Both the process and its result are abstract, and it is only by observing the behaviour of the model that the system builder can really know what it is like. Indeed it is not an unchanging entity, but depends for its structure at runtime on incoming data. In this sense it is more like a program than anything else, or, as we hinted above (section 6.1.2), a set of behavioural guidelines for a trainee.

Evaluators found the abstract nature of the model a hindrance and many wanted some graphical representation of it, although they were not at all clear as to what this might be like. The subjects were captivated by the idea of a 'model' and wanted this to be a real visual representation. It would certainly give better feedback when building the model if the system builder could 'see' it in a more concrete form.

There are limited options for representing such a task model. An unordered list of constraints, as we have now, is not particularly helpful. Algorithmic constraints can be ordered at least as far as the first subtask and small clumps of strictly ordered subtasks, but most models will be mixtures of algorithmic and whenever constraints. Should it then look like a program? This would be particularly unfriendly for the experts. Knowledge acquisition tools are, after all, supposed to remove the programming responsibility from the expert, not reintroduce it. However, there are ways of representing programs visually which are familiar to medical experts. Diagnostic flowcharts (e.g. [Essex 80], [BMJ 89]) and treatment protocols (eg [Jaffa 88]) are very like programming flowcharts. Making a full graphical representation of the task model is probably impossible but flowchart methods may provide some assistance. The potential of graphical programming languages is worthy of exploration.

6.2 Extensibility to other tasks

The methodology is of proven use when applied to the area of medical diagnosis and as such we can count it a success. However, the next step test for this methodology is to apply it to an unrelated problem-solving area.

6.2.1 Finding a suitable task

In demonstrating the TOMKAT toolkit to individuals in industry whose interests lay in modelling the question was often raised as to whether the technique could be applied to other tasks, especially those with industrial applications. The two main identifiable tasks with such applications are 'trouble-shooting', a type of diagnosis, and scheduling. Trouble-shooting tends to be a rather simple type of diagnosis, much simpler than medical diagnosis and certainly better understood. Scheduling, on the other hand, is viewed as something of a black art, much like medical diagnosis. In our evaluation, subjects with an AI background often referred to the differences between analysis tasks, like diagnosis, and synthesis tasks, like scheduling, noting that whereas the methodology we brought to bear on the diagnostic task was appropriate and successfully implemented with diagnosis, it might be much more difficult, perhaps impossible, with a synthetic task. Scheduling is an area where theory and practice are far apart. Little work has been done on modelling scheduling as a *task*, with the exception of some KADS modelling ([Balder *et al.* 92]) although this is as yet far from practical utility. This challenge, to model the scheduling task using this methodology, needs to be taken up.

6.2.2 Scheduling and humans

A schedule solution involves the allocation of resources and start times to activities subject to a set of constraints. Although it is often described as a constraint satisfaction problem, suggesting that any solution in which all constraints are satisfied are equally acceptable, in practice judgements are made regarding the relative quality of alternative solutions.

Scheduling is a hard task for humans to perform as it involves the retention and manipulation of large and shifting data-sets and multiple options. It is also an area where it is difficult to assess the utility of partial solutions. This is the sort of problem which benefits greatly from computational support. Such support would be of value in industrial scheduling applications as the scheduling task is one which is of crucial importance in the production industry. Even a small percentage improvement in a factory schedule can

result in significant improvements in production throughput and overall management. The creation of an environment for building and, importantly, testing and comparing task models of scheduling, would be of great interest to those involved across a wide range of manufacturing environments.

Recently there has been a trend in scheduling theory towards more 'interactive' schedulers ([Morton & Pintico 93]) which allow the human scheduler to have computational support rather than 'black-box' schedulers which produce solutions but whose methods are not open to inspection, the same situation we encountered with diagnostic systems in Chapter 1 section 1.1.2. Interactive schedulers which behave comprehensibly allow the user to interrupt the schedule, make for better explanations and allow the schedule builder control over the system's behaviour, many of the same advantages which we espoused for task modelling in medicine (Chapter 3). There is also an increasing demand for scheduling systems which allow incremental extension and change through modular system building ([Beck & Tate 94]) which our subtask methodology supports.

6.2.3 Applying the methodology to scheduling

We anticipate that the application of the methodology to the scheduling task would follow a similar programme to that used in this project. As this is a relatively untried domain much would depend on the responses of expert human schedulers.

Expert human schedulers would be consulted with a view to establishing the validity of the distinction between task and domain knowledge in the area. This involves identifying what they see as the scheduling task and the subtasks in scheduling that are used, establishing a task hierarchy if appropriate and outlining possible constraints on scheduling subtasks. Examples of scheduling subtasks which have already been suggested are

- choose-resource
- find-bottlenecks
- find-most-critical-resource
- choose-start-time

- sequence-operation

This phase would also focus on uncovering the task algorithms of automated schedulers and scheduling algorithms in the literature. This method was used successfully in the development of TOMKAT's diagnostic subtasks (see Chapter 2).

The output will feed into the construction of a library of scheduling algorithms, subtasks and constraints. As with diagnosis such primitives can be used to build 'rigid' task models of scheduling as described by expert schedulers and in the extant literature (such as [Currie & Tate 91], [Berry 91] [Beck 92]). If construction of rigid models is successful work can then proceed on developing a control mechanism to build more 'responsive' task models based on a constraint satisfaction mechanism as implemented in TOMKAT.

As with diagnosis, the expert human schedulers would be encouraged to participate in reverse-engineering to replicate their professed scheduling strategies. These task models would then be evaluated by being applied to test domains

There are two problems in carrying out this research. The first is the 'compiled knowledge' problem we referred to in Chapter 1. This is that experts may find it very difficult to identify their own task knowledge and even more difficult to replicate it. Experience with acquiring this task knowledge in medicine suggests that there are interview strategies that can assist this. It is also helpful to present sample subtasks with which experts can identify. The second problem is related to the difficulty of the scheduling task for human subjects. Industrial experts might not want to part with their knowledge unless there is some obvious gain for them. Unlike medicine, scheduling knowledge is not seen as 'public property' and a good human scheduling expert is a valuable resource. The obvious gain for such experts is that exposing their own strategies to their own inspection allows them to critique and improve these strategies in an objective, non-domain-dependent environment. A practical benefit would be the development of a framework for co-operative human and computer scheduling.

6.3 Conclusions

The initial aims of this project were to develop a methodology for analysing diagnosis and to implement that methodology as a tool allowing the exploration of diagnostic strategies and the building of diagnostic support systems.

Diagnostic strategies are not readily available for learners. They are difficult to get at from experts, teaching materials or software descriptions. Current knowledge acquisition tools are lacking in that there is not one which combines

- both domain and task model building abilities
- sufficient medicine-specific detail
- model retention
- simplicity and user-recognisability
- task model execution flexibility

in a system which is usable by domain experts. As far as we are aware there is no currently available tool which allows a domain expert to build system themselves in this manner. TOMKAT is innovative in that it

- is designed for use by domain experts themselves
- allows the expert to build both a domain and task model which will immediately run together as a complete diagnostic support system
- builds flexible task models which respond to incoming data

Our evaluation showed that experts can use the system and are able to both understand modelling concepts to significant extent and identify with the behaviour of the models they build. All our medical experts felt that it would be good for teaching purposes as diagnosis per se is not taught or only by routine protocols. TOMKAT would be ideal for allowing students to construct different diagnostic models to see how they behave.

The main outcomes of the project have been:

1. the development of a knowledge acquisition methodology for medical diagnosis with potential applicability to other problem-solving areas

2. the identification and formalisation of diagnostic subtasks, validated by domain experts and relevant to potential users' knowledge and experience

3. the construction of a tool with potential pedagogic value in medicine in an area (diagnostic strategy) which both trainers and trainees feel is severely lacking.

4. the construction of a knowledge acquisition tool for medicine which models both domain and task knowledge, is usable by experts and results in real systems.

What this project has achieved is to develop a methodology and associated software for modelling medical knowledge which has appeal to those in the domain. Focusing particularly on medical *problem-solving* knowledge enables the exploration of diagnosis as a task which can be taught explicitly and the building of medical support systems which behave diagnostically in a useful and appropriate manner. Finally, we wish to note that everyone who used the system *liked* it, and said so.

Bibliography

[Alpay 90] L. Alpay. *Modelling Medical Diagnostic Processes.* Unpublished PhD thesis, The Open University, Centre for Information Technology in Education, 1990. CITE PhD thesis number 9.

[Aylett 90] R. Aylett. Knowledge acquisition tools. *AIRING*, (10), 1990.

[Balder *et al.* 92] J. Balder, F. van Harmelan, and M. Aben. A KADS/(ML)2 model of a scheduling task. In J. R. Balder and J. M. Akkermans, editors, *Formal Methods for Knowledge Modelling in the CommonKADS methodology.* Netherlands Energy Research Foundation ECN, 1992.

[Balla 85] J. I. Balla. *The Diagnostic Process: a model for clinical teachers.* Cambridge University Press, Cambridge, 1985.

[Beck & Tate 94] H.A. Beck and A. Tate. Open planning, scheduling and constraint management architectures. *British Telecom Technology Journal*, 1994.

[Beck 92] H. Beck. Constraint Monitoring in TOSCA. In *Working Papers in AAAI Spring Symposium: Practical Approaches to Planning and Scheduling, Stanford*, 1992. Also available as a technical report (AIAI-TR-121).

[Berry 91] Pauline Berry. *A Predictive Model for Satisfying Conflicting Objectives in Scheduling.* Phd, Dept. of Computer Science, Strathclyde University, Glasgow, 1991.

[Bloomfield 86] B. P. Bloomfield. Capturing expertise by rule induction. *Knowledge Engineering Review*, 1(4), 1986.

[BMJ 89] BMJ. *Endocrine System: Clinical Algorithms.* British Medical Journal, Cambridge, 1989.

[Bramer 87] R. D. Bramer. Automatic induction of rules from examples: a critical analysis of the ID3 family of rule induction systems. In *Proceedings of the 1st European Workshop on Knowledge Acquisition*, Reading University, 1987. Reading University.

[Breuker & Wielinga 87] J. Breuker and B. Wielinga. Knowledge acquisition as modeling expertise: The KADS methodology. In *Proceedings of the 1st European Workshop on Knowledge Acquisition*, Reading University, 1987. Reading University.

[Chandrasekaran 86] B. Chandrasekaran. Generic tasks in knowledge based reasoning: High-level building blocks for expert system design. *IEEE Expert*, Fall, 1986.

[Chandrasekaran 88] B. Chandrasekaran. Generic tasks as building blocks for knowledge based systems: the diagnosis and routine design examples. *Knowledge Engineering Review*, 3, 1988.

[Clancey & Bock 88] W. J. Clancey and C. Bock. Representing control knowledge as abstract tasks and metarules. In L. Bolc and M. J. Coombs, editors, *Expert System Applications*. Springer Verlag, 1988.

[Clancey & Letsinger 84] W. J. Clancey and R. Letsinger. NEOMYCIN: reconfiguring a rule-based expert system for application to teaching. In *Readings in Medical Artificial Intelligence: the first decade*. Addison Wesley, 1984.

[Clancey 85] W. J.. Clancey. Heuristic classification. *Artificial Intelligence*, 27, 1985.

[Clancey 88] W. J. Clancey. Acquiring, representing, and evaluating a competence model of diagnostic strategy. In M. T. H. Chi, R. Glaser, and M. J. Farr, editors, *The Nature of Expertise*. Erlbaum, New Jersey, 1988.

[Currie & Tate 91] K.W. Currie and A Tate. O-Plan: the Open Planning Architecture. *Artificial Intelligence*, 51(1), 1991.

[Essex 80] B.J. Essex. *Diagnostic Pathways in Clinical Medicine*. Churchill Livingstone, Edinburgh, 1980.

[Evans & Patel 89] D. A. Evans and V. L. Patel, editors. *Cognitive Science in Medicine*. MIT Press, Cambridge, Mass., 1989.

[Feltovitch *et al.* 89] P. J. Feltovitch, R. J. Spiro, and R. L. Coulson. The nature of conceptual understanding in biomedicine: The deep structure of complex ideas and the development of misconceptions. In D. A. Evans and V. L. Patel, editors, *Cognitive Science in Medicine*. MIT Press, Cambridge, Mass., 1989.

[Forster 90] D. Forster. *Health Informatics in Developing Countries*. Unpublished PhD thesis, London School of Economics, Department of Information Systems, 1990.

[Fox & Krause 91] J. Fox and P. Krause. Symbolic decision theory and autonomous systems. In B. D. D'Ambrosio, P. Smets, and P. P. Bonissone, editors, *Uncertainty in Artificial Intelligence: Proceedings of the Seventh Conference*. Morgan Kauffman, 1991.

[Fox *et al.* 87] J. Fox, A. Glowinski, and M. O'Neil. The Oxford system of medicine: A prototype information system for primary care. In J. Fox, M. Fieschi, and R. Engelbrecht, editors, *AIME 87 European Conference on Artificial Intelligence in Medicine*. Springer Verlag, 1987.

[Fraser & Harrison 93] J. L. Fraser and I. W. Harrison. Modelling expertise using a belief network. In *Research and Development of Expert Systems: Proceedings of Expert Systems 93*. BHR Group Ltd. and British Computer Society Special Interest Group in Expert Systems, 1993.

[Funk *et al.* 87] M. Funk, R. D. Appel, Ch. Roch. D. Hochstrasser, Ch. Pellegrini, and F. Muller. Knowledge acquisition in expert system assisted diagnosis: A machine learning approach. In *AIME 87 European Conference on Artificial Intelligence in Medicine*. Springer-Verlag, 1987.

[Glowinski *et al.* 89] A. Glowinski, M. O'Neil, and J. Fox. Design of a generic information system and its application to primary care. In *AIME 89 Second Conference on Artificial Intelligence in Medicine*. Springer-Verlag, 1989.

[Godfrey 88] K. Godfrey. *Spot The Symptoms*. IPC Magazines Ltd., London, 1988.

[Hammond 80] K. R. Hammond. The integration of research in judgement and decision theory. Technical Report Report 226, University of Colorado, Institute of Behavioural Science, 1980.

[Hickman *et al.* 89] F. R. Hickman, J. L. Killin, L. Land, T. Mulhall, D. Porter, and R. M. Taylor, editors. *Analysis for Knowledge Based Systems: a practical guide to the KADS methodology*. Ellis Horwood, Chichester, 1989.

[Jaffa 88] A. Jaffa. *Sexually Transmitted Diseases: training manual*. Zimbabwe Ministry of Health, Harare, 1988.

[Karbach *et al.* 90] W. Karbach, M. Linster, and A. Voss. Models of problem-solving: One label - one idea? In B. Wielinga, J. Boose, B. Gaines, G. Schreiber, and M. van Someren, editors, *Current Trends in Knowledge Acquisition*. IOS press, Washington, 1990.

[Karel & Kenner 89] G. Karel and M. Kenner. Klue: A diagnostic expert system tool for manufacturing. *Intellinews*, 5(1), 1989.

[Kelly 55] G.A. Kelly. *The Psychology of personal Constructs.* Norton. New York, 1955.

[Kunz 88] J. C. Kunz. Model based reasoning in CIM. In M. D. Oliff, editor, *Intelligent Manufacturing: Expert Systems and the Leading Edge in Production Planning and Control.* Addison Wesley, 1988.

[Lanzola & Stefanelli 91] G. Lanzola and M. Stefanelli. A knowledge acquisition tool for medical diagnostic knowledge-based systems. In M. Stefanelli, A. Hasman, M. Fieschi, and J. Talmon, editors, *AIME 91 Third Conference on Artificial Intelligence in Medicine.* Springer-Verlag, 1991.

[Lanzola & Stefanelli 92] G. Lanzola and M. Stefanelli. A specialised framework for medical diagnostic knowledge-based systems. *Computers and Biomedical Research*, 25, 1992.

[Lanzola & Stefanelli 93] G. Lanzola and M. Stefanelli. Inferential knowledge acquisition. *Artificial Intelligence in Medicine*, 5, 1993.

[Leaper *et al.* 73] D. J. Leaper, P. W. Gill, J. R. Staniland, J. C. Horrocks, and F. T. De Dombal. Clinical diagnostic process: an analysis. *British Medical Journal*, 3, 1973.

[Machanik 88] P. Machanik. Design of medical education software as appropriate technology using artificial intelligence and software engineering. Technical Report 88-01, University of the Witwatersrand, Johannesburg, 1988.

[Morton & Pintico 93] T.E. Morton and D.W. Pintico. *Heuristic Scheduling Systems.* John Wiley, 1993.

[Pankhurst 79] R. J. Pankhurst. Medical diagnosis in developing countries. *Computers in Biology and Medicine*, 10:69 – 82, 1979.

[Parsaye 88] K. Parsaye. Acquiring and verifying knowledge automatically. *AI Expert*, May, 1988. Outline of AUTO-INTELLIGENCE tool.

[Pirnat *et al.* 89] V. Pirnat, I. Kononenko, T. Janc, and I. Bratko. Medical analysis of automatically induced diagnostic rules. In *AIME 89 Second Conference on Artificial Intelligence in Medicine.* Springer-Verlag, 1989.

[Quinlan 79] R. Quinlan. Discovering rules from large sets of examples: a case study. In D. Michie, editor, *Expert Systems in the Microelectronic Age.* Elsevier Science Publishers, Edinburgh, 1979.

[Ramsden *et al.* 89] P. Ramsden, G. Whelan, and D. Cooper. Some phenomena of medical students' diagnostic problem solving. *Medical Education*, 23(1):108 – 117, 1989.

[Rodolitz & Clancey 89] N. S. Rodolitz and W. J. Clancey. GUIDON-MANAGE: Teaching the process of medical diagnosis. In D. A. Evans and V. L. Patel, editors, *Cognitive Science in Medicine*. MIT Press, Cambridge, Mass., 1989.

[Rothenberg 89] J. Rothenberg. The nature of modelling. In L. E. Widman, K. A. Loparo, and N. R. Nielsen, editors, *Artificial Intelligence, Simulation and Modelling*. John Wiley and sons, Canada, 1989.

[Schijven *et al.* 89] R. A. J. Schijven, J. L. Talmon, E. Ermers, R. Penders, and P. J. E. H. M. Kitslaar. Machine learning as a knowledge acquisition tool: Application in the domain of the interpretation of test results. In *AIME 89 Second Conference on Artificial Intelligence in Medicine*. Springer-Verlag, 1989.

[Schreiber *et al.* 93] A. Th. Schreiber, G. van Heijst, G. Lanzola, and M. Stefanelli. Knowledge organisation in medical KBS construction. Technical Report GAMES-II/T1.1/UvA/PP/010/1.0, University of Amsterdam, 1993. AIM project A2034 GAMES-II.

[Shortliffe 76] E.H. Shortliffe. *Computer Based Medical Consultations: MYCIN*. American Elsevier, New York, 1976.

[Sox 88] H. Sox. *Medical Decision Making*. Butterworth, Boston, 1988.

[Wielinga *et al.* 92a] B. J. Wielinga, A. Th. Schreiber, and J. Breuker. KADS: a modelling approach to knowledge engineering. *Knowledge Acquisition*, 4(1), 1992.

[Wielinga *et al.* 92b] B. J. Wielinga, W. Van de Velde, G. Schreiber, and H. Akkermans. The commonKADS framework for knowledge modelling. Technical Report KADS-II/T1.1/PP/UvA/35/1.0, University of Amsterdam, 1992.

Appendix A

Underlying structure of TOMKAT

This appendix describes in outline the structuring of TOMKAT's code. The toolkit is written entirely in commonLISP and runs under KEE4. In all parts of the tookit implementation there is as little KEE specific functionality as possible. The KEE functionality is confined to displays and methods (slots) on objects of the same name which, when activated, call a function of the same name in an external lisp file. This is to ensure that the TOMKAT can be easily reimplemented outwith the KEE environment. Aside from the KEE knowledge bases associated with the tools there are five important code files 'outertool-functions', 'dmtool-functions', 'tm-tool-functions', 'utool-functions' and 'utilities'.

A.1 Outer Tool

TOMKAT is invoked by loading the file 'outertool-load.lisp' into KEE4. The outer model tool files consist of the outer tool knowledge base files and the outer tool functions file. Directions on how to use the outer tool are in Appendix C.

The outer tool functions file contains the functions delete-dmt, delete-tmt, delete-utool, invoke-dmt, invoke-tmt and invoke-utool. These provide the facilities to load and delete the task model tool, the domain model tool, or the user's tool.

A.2 Domain Model Tool

The domain model tool files consist of the domain model tool knowledge base files and the domain model tool functions file. Directions on how to use the domain model tool, build a domain model and save it are in Appendix C. kk/ph/design/versions/4iioct92.tex).

The domain model tool functions file contains:

Change functions: change-value, move-object, rename-object, rename-relation. For moving objects in the domain model around or renaming objects and slots.

Choose functions: choose-object, choose-relation, choose-subtree. For getting objects, slots or groups of objects out of the domain model tool library and copying them into the domain model you're building.

Display functions: display-dm, display-library. For showing the current hierarchical structure of the domain model, or part of it, or of the domain model tool library, or part of it.

Make functions: make-domain-model, make-new-object, make-new-relation. For making a new domain model, new objects in the domain model, in a hierarchical structure, or for making new slots on existing objects.

Remove functions: remove-object, remove-relation, remove-subtree, remove-value. For removing individual objects or slots or slot values or whole subtrees from the domain model being built.

Save functions: save-domain-model. For saving the built domain model into a knowledge base, currently always called dm-test.

A.3 Task Model Tool

The task model files consist of the task model tool knowledge base files and the task model tool functions file. Directions on how to use the task model tool, build a task model and save it are in Appendix C.

The task model tool functions file contains:

Algorithmic constraints: excludes, first-task, immediately-excludes, no-task-follows-itself, non-repeating, only-once, order, reject, strict-order. These are mainly ordering relationships between pairs of subtasks or ordering statements on single tasks. The retraction functions for these constraints are also included here.

How constraints: most-pathognomic. How constraints, which specify how a subtask must be effected, are only appropriate on choice subtasks (like choose-symptom) and the choice is limited at present to most-pathognomic and first.

Whenever constraints: all-general, empty, getting-big, none-general, not-empty, one-left, two-left. Whenever constraints specify the status that certain diagnostic objects must be in before a subtask can fire. These functions are all 'adjectives' like empty, all-general or getting-big.

Clearing functions: clear-constraints, reset-task-model, retract-algorithmic-constraints, retract-constraint, retract-how-constraint, retract-success-constraint, retract-whenever-constraint. These are for clearing and retracting constraints on subtasks in the task model being built.

Controller Functions: choose-another-task, critique, find-firable-tasks, find-rejected-tasks, find-unconstrained-tasks, find-unfirable-tasks, find-urgent-tasks, fire-task, get-number-constraints, justify-rejection, most-constrained, prioritise-tasks, show-firable-tasks, show-priority-tasks, show-unfirable-tasks, step-control. These are used in the task model tool for running and testing the model in a controlled manner and some are also used in the user's tool.

Explanation functions: explain, explain-question, explain-subtask. Simple functions that show what subtask is being executed or give a minimal description of the action of a subtask.

Save functions: Save-task-model saves the completed task model into a knowledge base called tm-test.

Setup functions: set-algorithmic-constraint, set-confirm-on-success, set-how-constraint, set-success-constraint, set-whenever-constraint. For setting up 'algorithmic', 'how' and 'whenever' constraints on subtasks and for setting the criterion of success and whether subtask confirm is fired once it is met.

Subtasks: change-path, choose-hypothesis, collect-a-symptom, confirm-hypothesis, differentiate, differentiate-2-hypothese, elaborate, group, ruleout. Subtasks are recognisable bits of the diagnostic process that are used to construct the task model.

Miscellaneous: There is also a ragbag of miscellaneous lowlevel functions used by subtasks, controlling functions and other bits of the task model tool. They are ask-user, calculate-current-symptoms, calculate-differential, check-focus, choose-symptom, elaborate-hypothesis, failure, find-unique-symptoms, get-hypotheses, get-symptoms, get-tasks, last-task, match-all-whenever-requirements, rejected, remove-false-symptom-hypotheses, remove-flags, remove-hypotheses-without-all-ks, success, unconstrained, urgent, why, why-not. These are often used by more than one higher function (e.g. choose-symptom is used by lots of things). Lowlevel functions which are very closely tied to higher operations are stored with these higher operations (e.g. the functions to restore different types of constraint are very tied to reset-task-model so are stored with that function in the 'clearing functions' area.)

A.4 User's Tool

The user tool files consist of the task model (tm-test) knowledge base files, the domain model (dm-test) knowledge-base files and the task model tool functions file. Directions on how to use the user tool are in Appendix C.

The utool-functions file contains the functions clear-user-system, diagnose, retract-symptom, retract-negative-symptom, retract-positive-symptom, select-dkb and select-task.

A.5 Utilities

There is a general file of utilities which are used by more than one tool. These are usually extensions to KEE or KEE shortcuts like my-single-choice-menu or get-link-type. They are used by all tools.

The utilities file contains the functions delete-leaves, descendants, find-leaves, find-non-leaves, find-objects-without-slot, has-not-slot, get-children, get-link-type, get-parents, leaf, leaf-descendants, my-single-choice-menu, remove-leaves, remove-non-leaves.

Appendix B

Evaluation

These are the contents of the introductory texts which were presented to the evaluation subjects. Details of the evaluation are in Chapter 5.

B.1 Introduction to concept

TOMKAT is a toolkit for building computer systems that do medical diagnosis. This document outlines the main ideas behind TOMKAT. Please read it *critically*, considering for each section:

- Does this idea make sense?
- Is it appealing?
- Could it be useful?
- Do I do anything like this in practice?

Two types of knowledge

We assume medical knowledge can be usefully considered to be of two types; knowing **what**, which we call *domain knowledge*, and knowing **how**, which we call *task knowledge*.

Domain knowledge is knowledge about the things and relationships that exist in an area, in this case about things like diseases and symptoms. This is the sort of medical knowledge that appears in medical textbooks. An example of domain knowledge might be 'a rash is a symptom of measles', or 'lactose intolerance is common amongst orientals'. Knowing about the relationships between diseases and symptoms is not much use to you unless you also know what to do with that knowledge; unless you also have some strategies to proceed from the identification of symptoms to recognition of an underlying disease.

Task knowledge is knowledge about how to do something, in this case how to do medical diagnosis. This is the sort of medical knowledge that is often gained only by 'doing it' or observing it being done (e.g. watching the specialist on ward rounds). An example of task knowledge might be 'identify which disease *type* you're dealing with before looking for the specific disease' or 'don't investigate more than one hypothesis at once'. Knowing how to do diagnosis is no use to you unless you also have something to diagnose; some diseases to recognise, some symptoms to observe.

In this evaluation we are most interested in *task knowledge*, and what we can do with it. We will not be considering domain knowledge much at all, so if you feel your medical experience is limited, or a bit rusty, this is not a problem.

Building a task model out of subtasks

We assume that the task of diagnosis can be described in terms of smaller subparts, or subtasks. For example, one simple-minded way of doing diagnosis might be to

- collect the presenting symptom and find all the diseases that might have that symptom
- choose one of those diseases and ask about another of its symptoms
- find all the diseases that have both the first symptom and the user's answer to the second symptom question
- continue thus until only one disease conforms to all the evidence

The above ordered list of diagnostic subtasks could serve as a (simple-minded, as we said) *model* of diagnosis. There are a multitude of other models of diagnosis which could be similarly defined.

Building a task model with constraints

There are two ways to build a task model out of subtasks. Firstly, you can specify exactly in which order the subtasks should be executed. The example above is like this. We call such an ordered list of diagnostic subtasks a *rigid* task model.

The other way to build a task model out of subtasks is to specify characteristics of each subtask which will guide when they can operate. This way you can build some 'sensitivity' into the model so that it responds to incoming data in an appropriate way. For example, consider a subtask called 'group' which groups diseases into types. It might be sensible to say 'group your possible diseases into types when they are all very specific diseases'. What you are doing here is putting a *constraint* on the 'group' subtask as to when it should operate.

Imagine the following scenario: A patient presents with a rash. It could be eczema or measles or flea-bites or nettlerash or an allergy or scabies or chickenpox or..... The

above constraint would guide you to group all the (many) possibilities into (perhaps) skin diseases, parasitic diseases, allergies and infectious fevers.

In fact, any statement about the ordering of the subtasks in a model can also be thought of as a constraint on when that subtask can operate. If you have a task called 'collect-symptom' which you always want to do first, you are actually placing a *constraint* on when that subtask should operate. This is what we call an 'algorithmic' constraint, because an ordered list of subtasks gives you an *algorithm* for diagnosis. The constraint we put on grouping above is a 'whenever' constraint, because you do the subtask *whenever* a certain state obtains.

Summary

Medical knowledge can be divided into knowledge of *how* to do diagnosis, **Task knowledge** and knowledge *what* things and relations obtain, **Domain knowledge.**

In building diagnostic support systems we think it is useful to *model* diagnosis by building out of diagnostic *subtasks.*

These subtasks can be either strictly ordered or we can place *constraints* on when they can operate so that the system you build decides at any point which subtask to fire based on the constraints you have set.

B.2 Introduction to product

TOMKAT is primarily a tool for building *models* in medicine. This evaluation is concerned with the tool for building a model of the diagnostic task itself, a model of how to do diagnosis.

The Subtasks

Models of diagnosis are assumed to be made up from *subtasks*, which are recognisable subparts of the diagnostic process. The subtasks implemented so far are:

change-path choose a general disease type and elaborate it

choose-hypothesis choose a disease to focus on

collect-a-symptom: select from the hierarchy of symptoms

confirm-hypothesis: ask about one of the disease' symptoms

differentiate: asks about a symptom that differentiates amongst the diseases

differentiate-2-hypotheses: asks about a symptom unique to one of two diseases

elaborate: looks up the lower-level types of diseases

group: looks up the more general types of diseases

ruleout: chooses a disease and asks about one of its unique symptoms

As it's difficult to imagine the effects of these rather abstract descriptions, it's probably better to see exactly how the subtasks behave when they are operating on some domain information. You will be able to test them out with the sofware.

Rigid models: 'algorithmic' constraints

As we described in the introductory document there are two ways of building a task model out of subtasks. The first is by building an ordered list of subtasks. In TOMKAT we do this by placing ordering *constraints* on subtasks. The available ordering, or 'algorithmic' constraints are:

excludes: subtask B can't fire if subtask A has fired

first: subtask A comes first

immediately-excludes: subtask B can't come immediately after subtask A

no-task-follows-itself: a constraint on all subtasks

non-repeating: subtask A can't follow itself

only-once: subtask A fires once only

order: subtask B comes after A

reject: subtask A can never fire

strict-order: subtask B comes immediately after A

Responsive models: 'whenever' constraints

You may remember in the introduction to the 'ideas' behind TOMKAT we spoke about the possibility of building a less rigid model, by using constraints on *when* subtasks might operate. An example we suggested was that it might be sensible to say 'group your possible diseases into types when the possibilities are all very specific'. The example given was:

A patient presents with a rash. It could be eczema or measles or flea-bites or nettlerash or an allergy or scabies or chickenpox or..... The above constraint would guide you to group all the (many) possibilities into (perhaps) skin diseases, parasitic diseases, allergies and infectious fevers.

The 'whenever' constraints are basically statements about what the status of your knowledge must be before you can fire a task. Such statements concern the status of **Diagnostic objects**. Diagnostic objects are basically just bags to hold things in. The most important diagnostic objects are the **Differential** and the **Focus.** The Differential is a bag that holds all the diseases that could possibly be the answer given what you know so far. The differential will get smaller the more information you put in. The Focus is a bag that holds that subset of the Differential that you are concentrating on at any given moment.

So the whenever constraint informally described above could be represented in TOMKAT's rather stilted English as 'group whenever differential none-general'. All whenever constraints are statements like this; the name of the subtask, the diagnostic object whose status is important, and the state it has to be in for that subtask to fire.

Critiquing

Since it's very time consuming to build up a diagnostic task model from scratch we have loaded in a 'default' model, a model that behaves in a more or less sensible way when doing diagnosis.

The easiest way to build a responsive task model is to 'critique' the behaviour of this model, altering its constraints when it asks the wrong questions, until it behaves as you would like.

Critiquing will step you through a diagnosis, choosing subtasks according to the preset constraints. At each point you can accept or reject the subtask chosen by the task model. If you reject it you can alter the model's constraints so that it will choose another subtask.

Summary

TOMKAT is a model building tool for diagnosis. A model of diagnosis consists of a lot of subtasks which can be rigidly ordered with the use of algorithmic constraints. Alternatively, a responsive task model can be constructed by putting constraints on when subtasks are fired.

A critiquing method has been implemented which allows the construction of a task model by 'tweaking' a default model until it behaves as you would wish.

B.3 Structured interview, concept evaluation

This is a selection of the type of questions asked. Not all of these questions would be asked as not all would be relevant to every subjects background and experience.

Demographic and evaluator's questions

1. medical experience: specialisation, practising?
2. computing experience: ai experience
3. Anything unclear from introduction?
4. Any further information needed?

Division between task and domain knowledge

1. Is it feasible to completely separate task and domain knowledge?
2. Do you understand/feel happy with the separation?
3. Do you think it's useful?
4. Do think you do it in practice? (medic-specific)
5. Do you see any problems with it?

Diagnosing and building a task model with subtasks

1. Does the idea of building a task model out of subtasks make sense?
2. Can you think of any subparts of diagnosis that could be used?
3. Are there any diagnostic methods you know you use in practice?
4. Do you use such a method when doing diagnosis? (medic-specific)
5. If you don't, do you think it would be a good idea?

Constraints, rigid and flexible models

1. Do you understand the notion of constraints?
2. Do you find these notions useful?
3. Do you do anything like this in practice?
4. Did you find the idea of rigid and flexible models appealing/familiar/useful?

5. Do you think 'rigid' or 'flexible' are what people use in practice or something in between?

6. Which parts of diagnosis might be 'rigidly' controlled (strictly ordered subtasks)?

General

1. Were you ever taught 'how to do diagnosis' independent of any specific domain? (medic-specific)

2. Do you think how you do diagnosis in [specialised area] is different from how colleagues in [other area] do it? (medic specific)

3. If so is why do you think this is? (medic-specific)

Appendix C

Using the System

In this appendix we describe the functionality of TOMKAT as it appears to the system builder. We detail how to build a task model using the task model tool and a domain model using the domain model tool, and how these combine in the end users' tool.

The system builder's interactions are almost entirely of the 'point and click' variety. The only typing in required is in the domain modelling tool, if the system builder wishes to create new objects and relationships rather than choosing existing ones. Every other aspect of model building is achieved through menu selection. This ensures that there can be no aberrant input from the system builder.

C.1 How to Use the Outer Tool

TOMKAT runs under Intellicorp's KEE (Knowledge Engineering Environment). The loadup file 'outertool-load.lisp' loads the outertool knowledge base which has functions for invoking and deleting the domain model tool, the task model tool and the users' tool. Figure C.1 shows the display of the outer tool.

Invoking the Task Model Tool

The 'invoke task model tool' button loads the task model tool knowledge base, the utilities file, task model tool functions and the users' tool functions (for testing the task model).

Invoking the Domain Model Tool

The 'invoke domain model tool' button loads the dmtool knowledge base, the dmtool functions file and the utilities file.

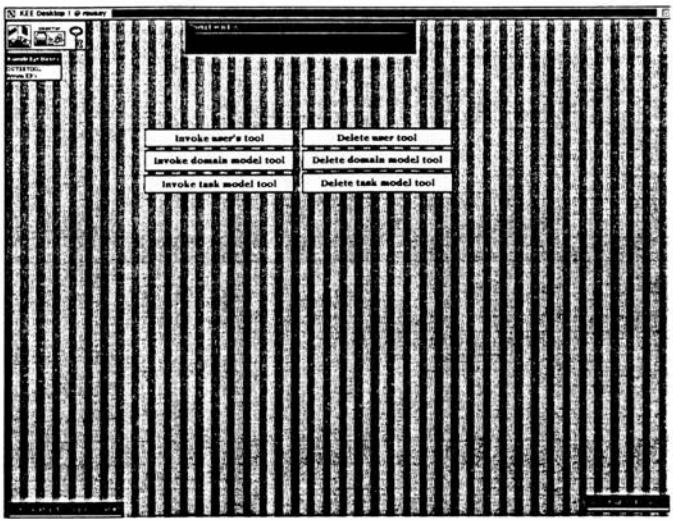


Figure C.1: Outer tool display

Invoking the Users' Tool

The 'invoke-user-tool' button loads the task model knowledge base tm-test, the domain model knowledge base dm-test, the task model tool functions file, the utilities file and the users' tool functions file.

C.2 How to Build a Task Model

The task model tool consists of mechanisms for 'doing' diagnosis interactively, for putting constraints on the firing of subtasks, and for specifying what counts as diagnostic success. As well as using these facilities unaided the system builder can also use them in a 'critiquing' mode, whereby the behaviour of a default task model is altered until it conforms to what is required. The built task model can be tested and saved. Figure C.2 shows diagrammatically the steps involved in building a task model.

When the 'invoke-task-model' button is clicked in the outer tool the options displayed in figure C.3 appear.

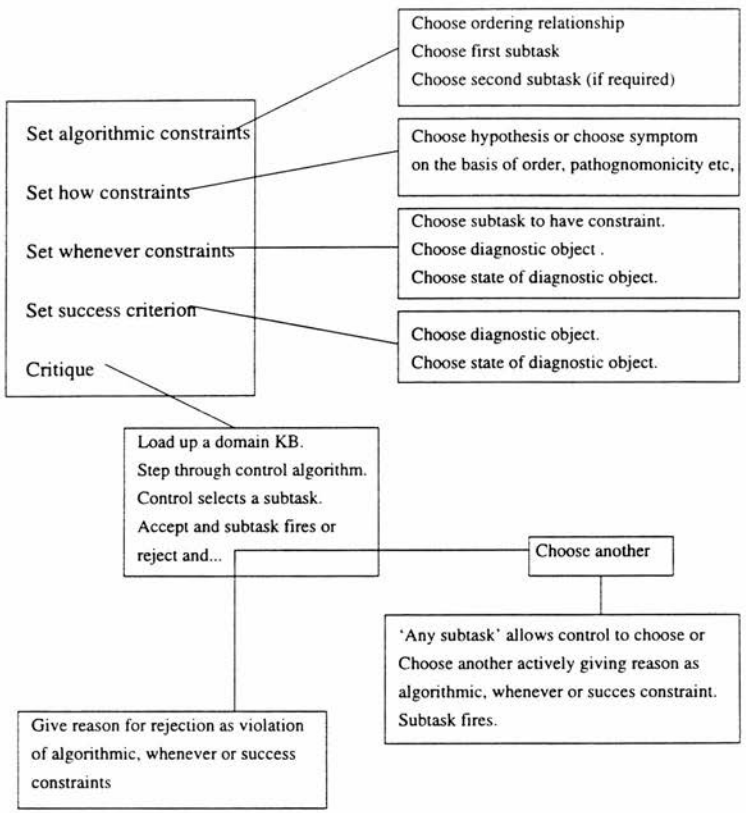


Figure C.2: Building a task model

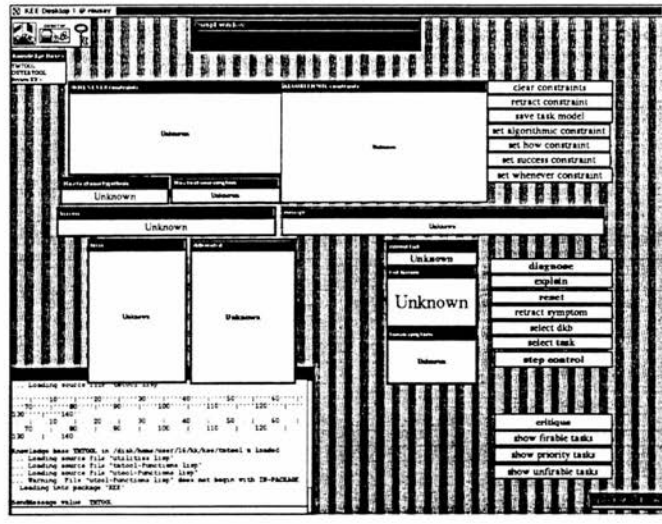


Figure C.3: Task model tool display

Seeing how diagnosis works

The middle set of right-hand buttons is a set of facilities for 'doing' diagnosis, either interactively by selecting subtasks or using a pre-prepared task model.

1. **Diagnose:** Sets the controller in motion. Continually chooses a subtask to fire (using the algorithm described in Chapter 4 section 4.4.3) until it meets its success criterion. It then confirms whatever is left, if the 'confirm-on-success' option has been set.

2. **Explain:**
   Explain question: gives the name of the subtask currently being attempted.
   Explain subtask: puts a minimal description of the action of the current subtask into the message box.

3. **Reset:** Clears out what has been entered by the user this session in the diagnostic objects (focus, differential, known-symptoms, task-history etc.) and restores the original task model constraints which may have changed through the diagnosis.

4. **Retract symptom:** Both positive and negative symptoms can be retracted. Retracting a symptom causes the diagnostic objects to be recalculated to their previous state.

5. **Select DKB:** For loadin a domain knowledge base. There are some ready made domain models for the subtasks to work on including one covering childhood diseases (based on [Godfrey 88]) and one covering sexually transmitted diseases (based on [Jaffa 88]). These were constructed using the domain modelling tool.

6. **Select task:** At any point in a diagnosis the user can select a subtask to fire. Subtasks are small, recognisable bits of the diagnostic process. Subtasks currently implemented, whose operations are detailed in Chapter 4 section 4.4.1, are

   - change-path
   - choose-hypothesis
   - collect-a-symptom
   - confirm-hypothesis
   - differentiate
   - differentiate-2-hypotheses
   - elaborate
   - focus-on-differential
   - group
   - ruleout

   The behaviour of individual subtasks can be examined with 'select-task'. The selected subtask will operate on the currently loaded domain knowledge base. For example, the subtask 'collect-a-symptom', the most basic task of diagnosis, displays a cascading menu of the symptom hierarchy from the current domain model. The system builder can choose a symptom object from any level of the hierarchy. Figure C.4 shows the result of selecting the 'collect-a-symptom' subtask and choosing 'fever' from the cascading menu which displays the symptom hierarchy in the domain knowledge base of childhood diseases.

7. **Step control:** Step control is for doing diagnosis with a pre-prepared task model in a controlled manner by stepping through each cycle of the control algorithm to see which subtask it chooses. Subtasks are again chosen on the basis of their constraints being satisfied.

Setting up the model

The top right hand set of buttons in the task model tool is for setting up the task model. This consists of setting the three types of constraints and the criterion of success.

1. **Clear-constraints:** removes all set constraints from the subtasks.

2. **Retract constraint:** Any constraint can be retracted in the same manner that it was set up e.g. retracting a 'whenever' constraint requires the builder to specify the subtask, the diagnostic object and the object state.

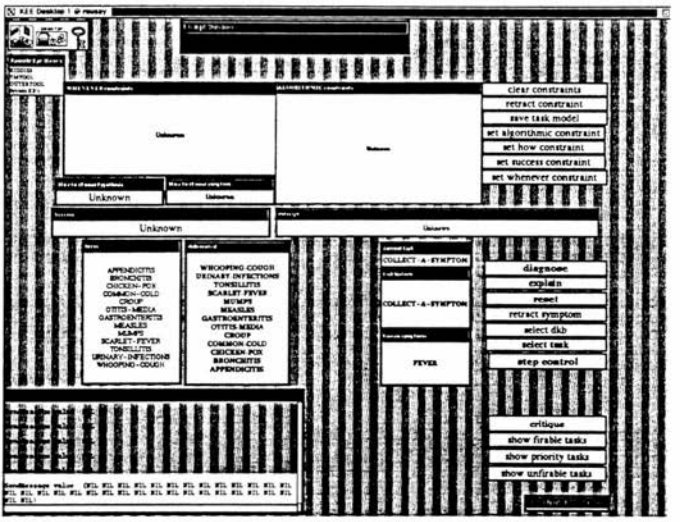


Figure C.4: Result of 'collect-a-symptom' selecting 'fever' on the 'kiddies' domain model

3. **Saving the task model** The saving function saves the task model as a KEE knowledge base called 'tm-test'. What gets saved as the task model is
   - the **Diagnostic Objects:** with contents at the time of saving
   - the **Subtasks** with the constraints set.
   - the **Task-model** and **User** objects.
4. **Setting Algorithmic constraints:** 'Algorithmic' constraints are generally ordering constraints on subtasks. The ordering constraints currently implemented are
   - excludes
   - first-task
   - immediately-excludes
   - no-task-follows-itself
   - non-repeating
   - only-once
   - order
   - reject
   - strict-order

Set-algorithmic-constraint prompts for the ordering function to be set and which subtask(s) are involved (e.g. that 'elaborate' is not to immediately follow 'group'). Figure C.5 shows the endpoint of this process.

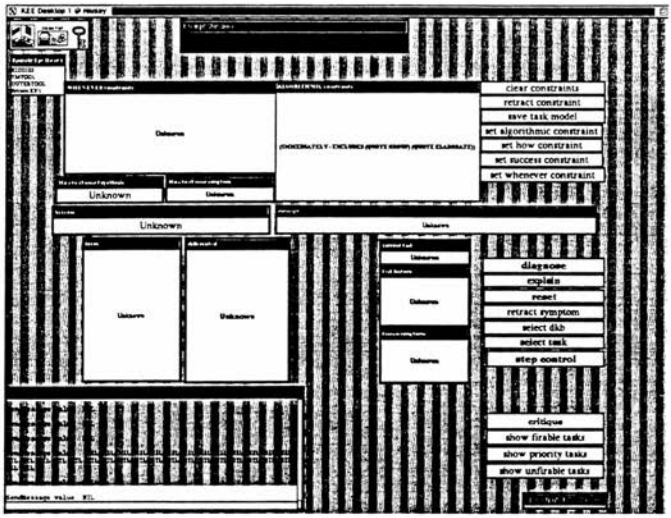


Figure C.5: Setting an 'algorithmic' constraint

5. **Setting How constraints:** 'How' constraints affect how a subtask operates. How constraints are only available on the two choice subtasks; choose-symptom and choose-hypothesis. There are at present only two options, most-pathognomic and first. **Set-how-constraint** asks whether hypothesis or symptom choice is affected and which choice method is to be used (e.g. that choose-symptom is to choose the most-pathognomic symptom). Figure C.6 shows the endpoint of this process.

6. **Setting the success criterion:** Success is the value of the slot *success-constraint* on the **task-model** object. It is set exactly as a whenever-constraint is set: by selecting a diagnostic object and then selecting the state that object must be in in order to halt the diagnostic process (e.g. that the 'differential' has 'one-left'). The same state set is available to the success criterion as to 'whenever' constraints. Figure C.7 shows the endpoint of this process.

   It is also possible to set whether diagnosis will call the subtask 'confirm-hypothesis' as its last action by selecting 'confirm-on-success' after selecting 'set-success-constraint'.

7. **Setting Whenever constraints:** 'Whenever' constraints are used by the controller to decide which subtasks can be fired. 'Whenever' constraints implement requirements which must be met before a subtask can be fired. Such a requirement

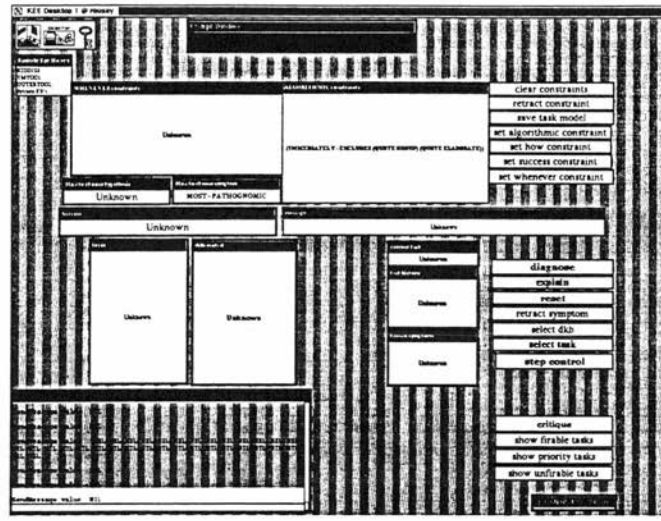


Figure C.6: Setting a 'how' constraint

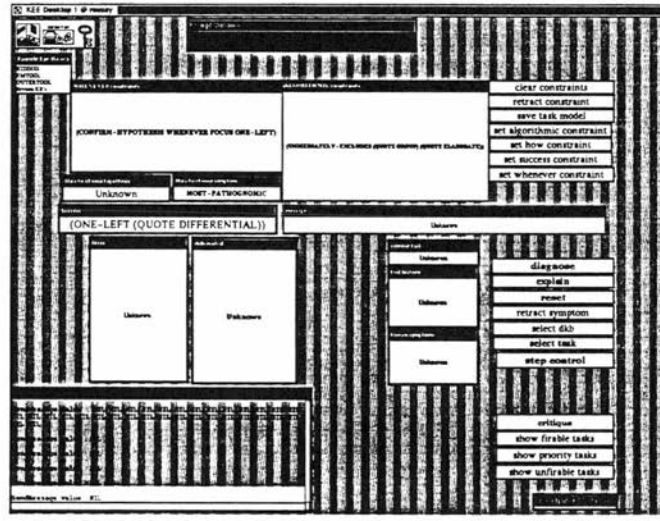


Figure C.7: Setting the success criterion

consists of the subtask, the object that needs to be in a certain state, and the state the object must be in before the subtask can fire. The object states currently offered are

- all-general
- empty
- getting-big
- none-general
- not-empty
- one-left
- two-left

Set-whenever-constraint prompts for the affected subtask, the diagnostic object whose status influences whether it can fire, and the status of that diagnostic object which will allow it to fire (e.g. the subtask 'confirm-hypothesis' is to fire whenever the focus object has one thing left in it). Figure C.8 shows the endpoint of this process.

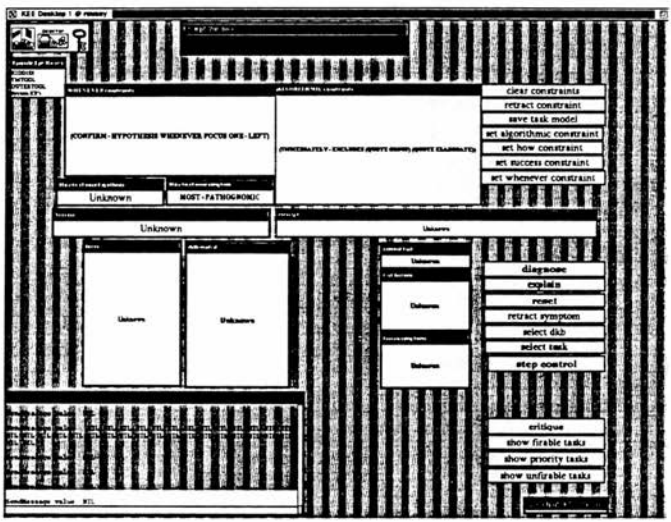


Figure C.8: Setting a 'whenever' constraint

Critiquing

The bottom right hand set of buttons in the task model tool is for critiquing the model. Several extra facilities are available to assist in the critiquing process by giving some

insight into the status of subtasks.

1. **Critique:** In this mode of task model building the controller chooses a subtask according to the constraints in a default task model. The system builder accepts or rejects the choice of subtask. If they accept it the subtask is fired. If they reject it they can choose another subtask (including 'any subtask'), and give a reason for the choice as a constraint, or give a reason for rejection as a constraint which has not been met. These constraints will then be added on to the current task model. The system builder can also, of course, retract any constraints from the model.

2. **Show firable tasks:** Firable subtasks are non-rejected and urgent subtasks whose constraints are all satisfied, or which are unconstrained, at the current state of diagnosis.

3. **Show priority tasks:** Non-rejected subtasks are prioritised thus: urgent subtasks, subtasks with constraints satisfied, unconstrained tasks.

4. **Show unfirable tasks:** Unfirable subtasks are those whose constraints are not met or which have been rejected at the current state of diagnosis.

Figure C.9 shows the use of the 'show unfirable tasks' option during the critiquing af a partially completed task model.

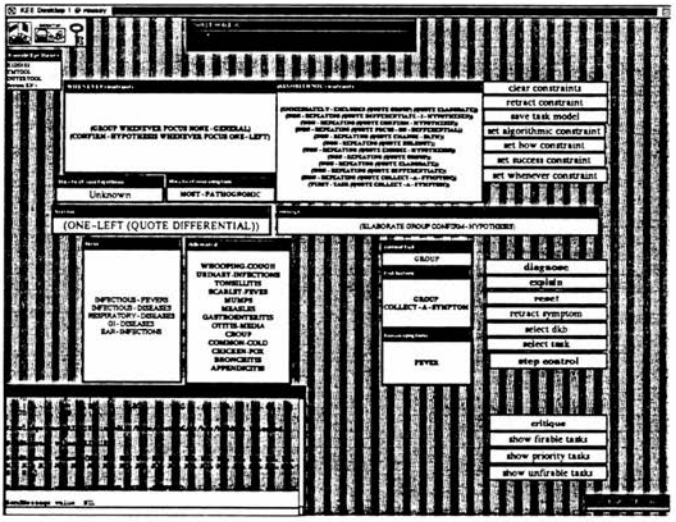


Figure C.9: Critiquing: showing unfirable subtasks

C.3 How to Build a Domain Model

The domain model tool is a simple laddering type tool for constructing disease and symptom object hierarchies, with links between diseases and symptoms. There is also a disease and symptom hierarchy library from which objects, relations and subtrees can be borrowed. The completed model can be saved. Figure C.10 shows diagrammatically the main steps involved in building a domain model.

Figure C.11 shows how the domain model tool displays initially when invoked from the outer tool.

Making things

1. **Make-new-domain-model:** 'Make-new-domain-model' creates a new knowledge base called 'dm-test'. Any domain model must at least contain the seed high level objects 'symptoms' and 'diseases' so the new model has these. Figure C.12 shows how a new domain model is 'seeded'. All new, or chosen, objects must be of these seed types and they have characteristic slots (relations) which are inherited by any descendant. This is so that the subtasks in the task model have something they know about to work on.

2. **Make-new-object:** 'Make-new-object' prompts for the name of the object. The system builder is then led through the existing domain hierarchy to choose the new object's immediate parent, (which can be a new object whose name and parent will be prompted for in the same manner). Figure C.13 shows how a new domain object is added to the hierarchy.

3. **Make-new-relation:** This prompts for the name of the new relation and then leads the system builder through the domain hierarchy to identify the object which should have this slot attached to it. Values of slots are inserted using the 'fill-in' facility.

Choosing things

1. **Choose-object** 'Choose-object' leads the system builder through the domain model object hierarchy. Once the object is selected the new parent object in the domain model being built is identified by a similar traverse through its hierarchy.

2. **Choose-relation** There is a subtree in the domain model library of relations, 1-place (qualities) 2-place and n-place. Once the relation has been selected 'choose-relation' leads the system builder through the domain model hierarchy under construction to identify which object it should be attached to.

3. **Choose-subtree** The domain model library consists of a taxonomic medical hierarchy, parts of which can be borrowed and inserted into the domain model hierarchy under construction. Once the subtree has been selected 'choose-subtree' leads the

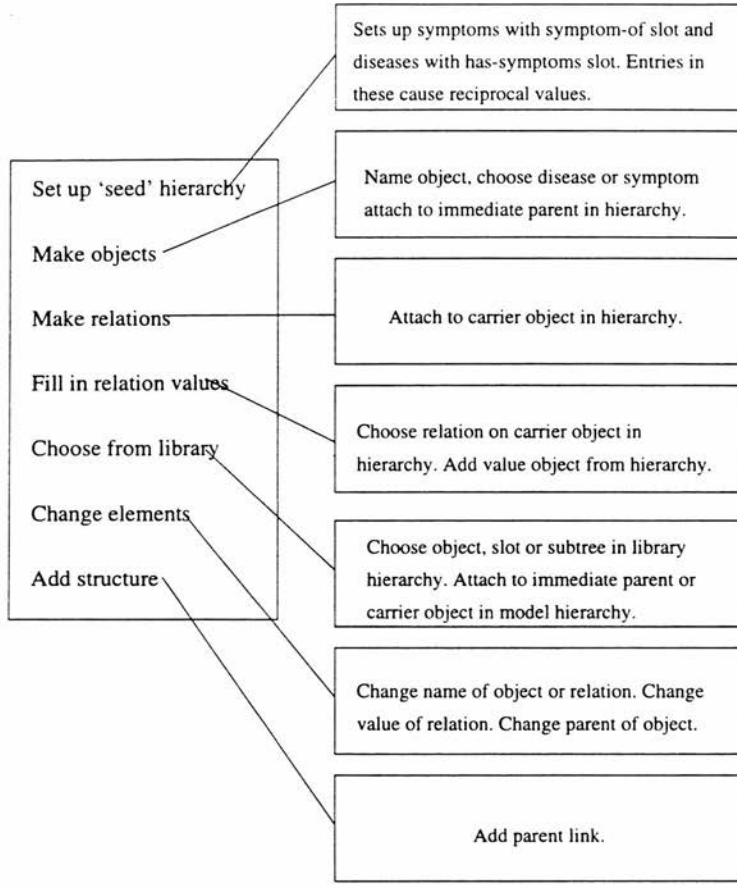


Figure C.10: Building a domain model

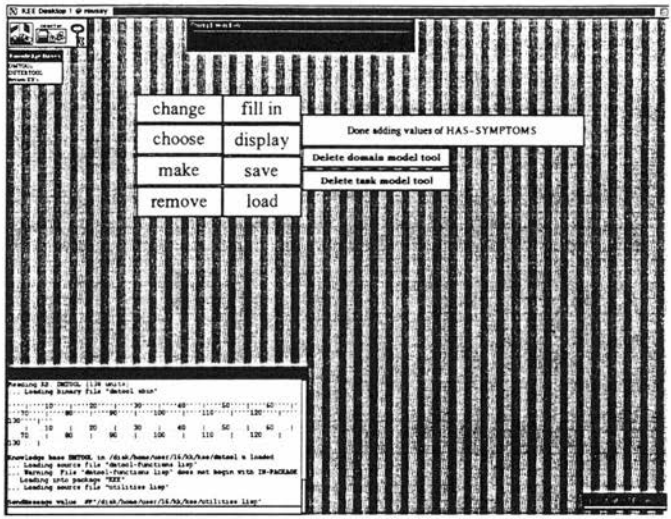


Figure C.11: Domain model tool display

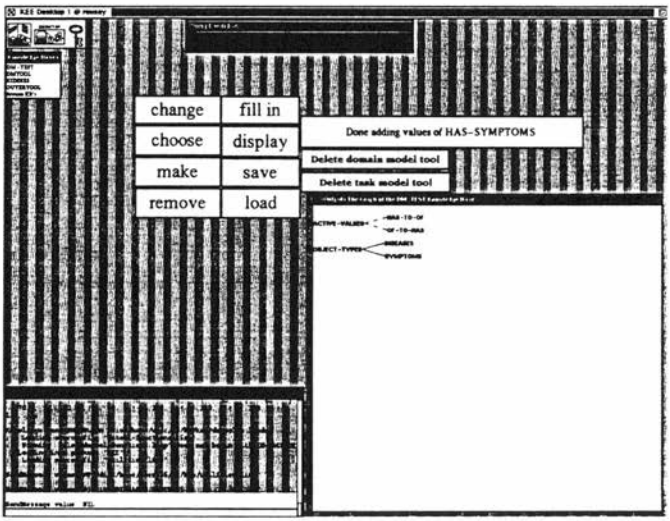


Figure C.12: New 'seeded' domain model

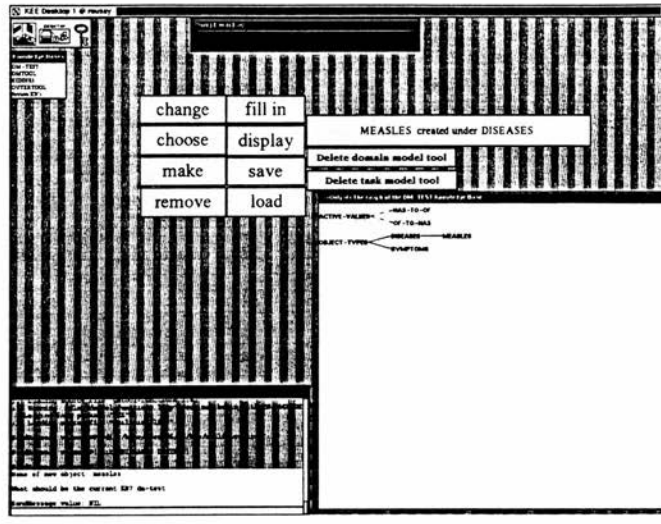


Figure C.13: Creating a new domain object

system builder through the domain model hierarchy to identify the new parent of the subtree. Figure C.14 shows part of the domain model library hierarchy.

Changing things

1. **Move-object:** Leads the user through the hierarchy to identify the object to be moved and the new parent. Sets up the new subclass link and deletes the old.
2. **Rename-object:** Leads the user through the hierarchy to identify the object to be renamed. Prompts for the new name and renames the object.
3. **Rename-relation:** Leads the user through the hierarchy to identify the object with the relation, allows the user to select the relation to be renamed and prompts for the new relation name. Deletes the old relation and adds the new one.

Display

1. **Display domain model:** Redisplays the current dm-test knowledge base.
2. **Display library** Displays the domain model tool library, a taxonomic hierarchy of diseases and symptoms.

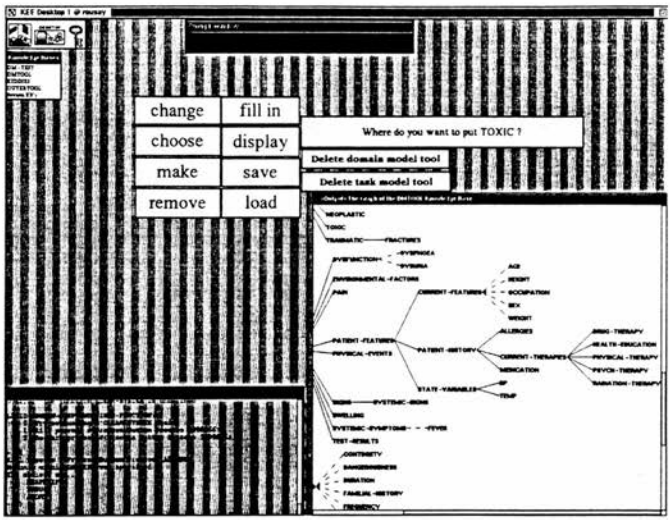


Figure C.14: Part of domain model library hierarchy.

'Fill in'

For filling in the values of relations (slots). Note that there are active values attached to the slots *has-symptoms* and *symptom-of* on the seed objects which ensures that complementary relations will be set up whenever a value is entered (Entering 'syphilis has-symptoms ulcers' causes the complementary 'ulcers symptom-of syphilis' to be set up). Leads the user through the hierarchy to identify the object, allows selection of the relation and leads the user through the hierarchy to find the name of the object which is to be a value or allows a new value to be entered.

Loading a model

For loading a partially-completed model. 'Load' Prompts for the name of a model being worked on and loads its knowledge base in as the current domain knowledge base.

Removing things

1. **Remove object:** Leads the user through the hierarchy to identify the object to be removed and removes it.
2. **Remove relation:** Leads the user through the hierarchy to identify the object with the relation, allows selection of the relation and removes it.

3. **Remove subtree:** Leads the user through the hierarchy to identify the toplevel node of the subtree. Removes that node and all its descendants.

4. **Remove value:** Leads the user through the hierarchy to identify the object, allows selection of the relation and the value and removes the value.

Save

Saves the current domain knowledge base as knowledge base dm-test.

C.4 How to Use the Users' Tool

The users' tool consists of these two models acting together, with a more limited functionality (only for using, not changing models). The users' main interaction with the built system is to fire it up and answer the subsequent questions by selecting from menus. Figure C.15 shows how the users' tool displays initially when invoked from the outer tool. The facilities available to the end user are a limited set of those in the task modelling tool, namely

- Diagnose
- Explain
- Reset
- Retract symptom
- Select dkb
- Select task

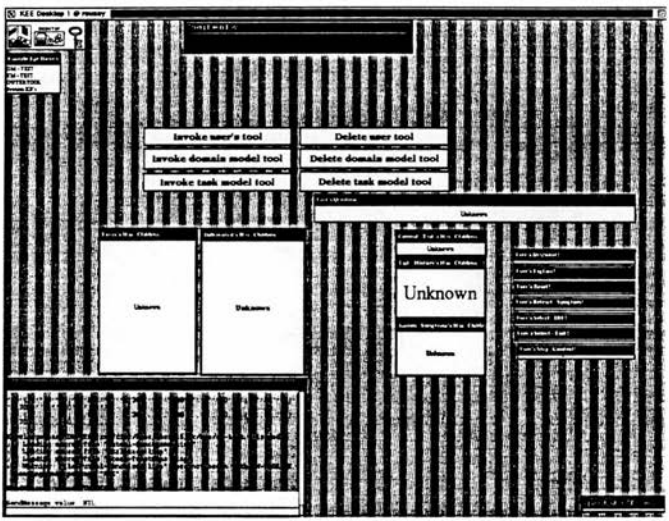


Figure C.15: Users' tool display

at Abnormal Carrier Frequencies. *J. comp. physiol.*, 146:361–378, 1982.

[17] B. Webb. Using robots to model animals: a cricket test. *Robotics and Autonomous Systems*, 16:117–134, 1995.